Acceleration of Bioinformatics Workflows on Shared Clusters

Ferran Badosa Galí

Universitat Autònoma
de Barcelona

Escola d'Enginyeria
Departament d'Arquitectura de Computadors i Sistemes Operatius

Acceleration of Bioinformatics Workflows on Shared Clusters

Memòria presentada per en **Ferran Badosa Galí** per optar al grau de Doctor per la Universitat Autònoma de Barcelona, sota la direcció de la Catedràtica Ana Ripoll Aracil, el Dr. Antonio Espinosa Morales i el Dr. Gonzalo Vera Rodríguez.

Cerdanyola del Vallès, setembre de 2018

Acceleration of Bioinformatics Workflows on Shared Clusters

Memòria presentada per en **Ferran Badosa Galí** per optar al grau de Doctor per la Universitat Autònoma de Barcelona. Aquest treball ha sigut desenvolupat en el Departament d'Arquitectura de Computadors i Sistemes Operatius de l'Escola d'Enginyeria, dins del programa de Doctorat en Informàtica, sota la direcció de la Catedràtica Ana Ripoll Aracil, el Dr. Antonio Espinosa Morales i el Dr. Gonzalo Vera Rodríguez.

Dr. Antonio Espinosa Morales
Director

Dra. Ana Ripoll Aracil
Directora

Dr. Gonzalo Vera Rodríguez
Director

Ferran Badosa Galí
Autor de la Tesi

Cerdanyola del Vallès, setembre de 2018

Index of Contents

Index of Figures

Index of Tables

Abstract

Shared clusters with multi-socket multi-core nodes have become common platforms to execute bioinformatics workflow applications. In order to harness the power offered by clusters, meet in so far as possible time or cost conditions, RMS on clusters must properly allocate the applications to resources. To do so, they must account for the particularities of bioinformatics applications, which unlike most other applications typically executed on clusters, can yield different resource usages from one execution to the next one, depending on the values given to their configuration parameters and the characteristics of the biological dataset analyzed. Current RMS on clusters fail to account for these particularities, and allocate bioinformatics applications to shared cluster resources regardless of the actual needs of applications. As a result, cluster resources are wasted or low utilized, and the performance of applications drops.

To tackle these issues, in this thesis we introduce a History-based Resource Manager (HBRM) for bioinformatics workflow applications in shared clusters. The goal of the HBRM is to determine the adequate combinations of resources that bioinformatics workflow applications need, in order to improve their performance in shared clusters obtained with current RMS. The HBRM is composed of different blocks: an Application characterization block, a Multivariate Regression Prediction, and a Scheduler multicriteria. First, we conduct a series of experiments in a cluster to characterize bioinformatics applications, monitor their performance and store it in a historical database. Second, we designed a Multivariate Regression predictor, which based on historical performance information of applications' previous runs, estimates the resources applications submitted to the cluster need. Third, we developed a Scheduler Multicriteria, which based on performance predictions, and the makespan slowdown of multiprogrammed nodes, schedules bioinformatics applications queued on the cluster. In the Scheduler Multicriteria, we designed three different scheduling algorithms for bioinformatics applications: the Slowdown-Aware (SA) scheduling algorithm, the Biobackfill scheduling algorithm, and the Slowdown-Aware File-Placement (SAFP) algorithm.

To validate the HBRM with all its scheduling algorithms, we conduct a series of experiments. We process a series of multiworkflows in different clusters, with resources being managed by the HBRM, and other state of the art scheduling algorithms. With the experiments, we prove that the HBRM can improve the makespan, resource utilization and efficiency of a queue of bioinformatics workflow applications obtained with the other state of the art algorithms. In the first validation experiments, we prove that the HBRM with the SA algorithm can improve the average workflow makespan by up to 34%, the average resource utilization by 86%, and the average resource efficiency by 96% obtained with SLURM's FCFS on clusters. In the second validation experiments we prove that the HBRM with the Biobackfill algorithm can improve

the average workflow makespan obtained with Bestfit and Firstfit algorithms by 8.55%. In the third validation experiments we prove that the HBRM with the SA algorithm can improve the makespans of Epigenomics, Montage, and Sipht multiworkflows obtained with Min-Min and Max-Min on large clusters by 43%. Also, we prove on large clusters and with the same workloads that the SAFP algorithm can improve the multiworkflow makespan of the SA algorithm by 88%, and the multiworkflow makespan of Min-Min and Max-Min by 93%.

Abstract

Els clústers compartits amb multiples sockets i multiples cores han esdevingut plataformes populars on executar applicacions de workflows bioinformàtics. Per tal d'aprofitar la potència de còmput que ofereixen els clústers, i complir en la mesura del possible amb les restriccions de cost o temps, els RMS dels clústers han d'assignar els recursos de manera correcta. Per tal de fer això, els enfocaments seguits per els RMS han de tenir en compte les particularitats de les aplicacions bioinformàtiques, que a diferència de moltes altres aplicacions comunment executades en clústers, poden adoptar diferents usos de recursos d'una execució a la següent, segons els valors atorgats als paràmetres de configuració de les aplicacions, i les característiques dels datasets analitzats. Actualment, els RMS dels clústers no consideren aquestes particularitats, i assignen aplicacions bioinformàtiques a recursos de clusters compartits obviant les necessitats reals de les aplicacions. Consequentment, els recursos dels clústers són poc utilitzats, i el rendiment de les aplicacions disminueix dràsticament.

Per afrontar aquests problemes, en aquesta tesi presentem el History-based Resource Manager (HBRM) per aplicacions de workflows bioinformàtics en clústers compartits. L'objectiu de l'HRBM és determinar l'adequada combinació de recurusos que les aplicacions necessiten, per tal de millorar-ne el rendiment assolit amb els RMS actuals. El HBRM està format per diversos blocs: un bloc de Caracterització d'Aplicacions, un Predictor de Regressió Multivariant, i un Planificador multicriteri. Primer, hem dut a terme una serie d'experiments en un clúster per caracteritzar aplicacions bioinformàtiques, monitoritzar-ne el rendiment i emmagatzemar-lo en una base de dades històrica. Segon, hem dissenyat un Model de Predicció Regressió Multivariant, que basat en prèvies execucions de les aplicacions en el clúster, estima els recursos que les aplicacions necessiten. Tercer, hem desenvolupat un Planificador Multicriteri que basat en prediccions de rendiment, i el makespan slowdown dels nodes multiprogramats, planifica la cua d'aplicacions bioinformàtiques del clúster. Dins del Planificador Multicriteri, hem dissenyat 3 algoritmes de planificació differents: l'algoritme de planificació Slowdown-Aware (SA), l'algoritme de planificació Biobackfill, i l'algoritme de planificació Slowdown-Aware File-Placement (SAFP).

Per validar l'HBRM amb tots els algoritmes de planificació, hem dut a terme una serie d'experiments. Hem processat una serie de multiworkflows en diversos clústers, amb els recursos gestionats tan per l'HBRM com per altres algoritmes de l'estat de l'art. Amb els experiments, hem demostrat que l'HBRM pot millorar el makespan, l'ús de recursos i l'eficiència d'una cua de workflows d'aplicacions bioinformàtiques obtinguda amb altres polítiques de l'estat de l'art. En la primera ronda d'experiments de validació, hem demostrat que l'HBRM amb l'algoritme SA, pot millorar el workflow makespan mitjà fins un 34%, l'ús de recursos fins un 86%, i la eficiència

mitjana dels recursos fins un 96% en comparació amb les mateixes mètriques obtingudes amb FCFS de SLURM en clústers. En la segona ronda d'experiments de validació hem demostrat que el HBRM amb l'algoritme Biobackfill pot millorar el makespan mitjà dels workflows en vers Firstfit i Bestfit un 8.55%. En la tercera ronda d'experiments de validació demostrem que el HBRM amb l'algoritme SA pot millorar els makespans d'Epigenomics, Montage i Sipht multiworkflows obtinguts amb Min-Min i Max-Min en clústers amb un gran número de nodes un 43%. A més, hem demostrat que en clústers amb un gran número de nodes i amb les mateixes càrregues de treball, que l'algoritme SAFP pot millorar el multiworkflow makespan de l'algoritme SA un 88%, i el multiworkflow makespan de Min-Min i Max-Min un 93%.

Abstract

Los clústers compartidos con múltiples sockets y multiples cores se han convertido en plataformas populares donde ejecutar aplicaciones de workflows bioinformáticos. Para aprovechar la potencia de cómputo que ofrecen los clústers, y cumplir en la medida de lo posible con las restricciones de coste o tiempo, los RMS de los clusters han de asignar los recursos de manera correcta. Para hacer esto, los enfoques seguidos por los RMS deben tener en cuenta las particularidades de las aplicaciones bioinformáticas, que a diferencia de muchas otras aplicaciones comúnmente ejecutadas en clústers, pueden adoptar diferentes usos de recursos de una ejecución a la siguiente, según los valores otorgados a los parámetros de configuración de las aplicaciones, y las características de los datasets analizados. Actualmente, los RMS de los clústers no consideran estas particularidades, y asignan aplicaciones bioinformáticas a recursos de clusters compartidos obviando las necesidades reales de las aplicaciones. Consecuentemente, los recursos de los clústers son poco utilizados, y el rendimiento de las aplicaciones disminuye drásticamente.

Para afrontar estos problemas, en esta tesis presentamos el History-based Resource Manager (HBRM) para aplicaciones de workflows bioinformáticos en clústers compartidos. El objetivo del HRBM es determinar la adecuada combinación de recurusos que las aplicaciones necesitan, para mejorar el rendimiento alcanzado con los RMS actuales. El HBRM está formado por varios bloques: un bloque de Caracterización de Aplicaciones, un Predictor de Regresión Multivariante, y un Planificador Multicriterio. Primero, hemos llevado a cabo una serie de experimentos en un clúster para caracterizar aplicaciones bioinformáticas, monitorizar el rendimiento y almacenarlo en una base de datos histórica. Segundo, hemos diseñado un Modelo de Predicción Regresión Multivariante, que basado en previas ejecuciones de las aplicaciones en el clúster, estima los recursos que las aplicaciones necesitan. Tercero, hemos desarrollado un Planificador Multicriterio que basado en predicciones de rendimiento, y el makespan slowdown de los nodos multiprogramados, planifica la cola de aplicaciones bioinformáticas del clúster. Dentro del Planificador Multicriterio, hemos diseñado 3 algoritmos de planificación differents: el algoritmo de planificación Slowdown-Aware (SA), el algoritmo de planificación Biobackfill, y el algoritmo de planificación Slowdown-Aware File-Placement (SAFP).

Para validar el HBRM con todos los algoritmos de planificación, hemos llevado a cabo una serie de experimentos. Hemos procesado una serie de multiworkflows en varios clústers, con los recursos gestionados tanto por el HBRM como por otros algoritmos del estado del arte. Con los experimentos, hemos demostrado que el HBRM puede mejorar el makespan, el uso de recursos y la eficiencia de una cola de workflows de aplicaciones bioinformáticas obtenida con otras políticas del estado del arte. En la primera ronda de experimentos de validación, hemos demostrado que la HBRM con el algoritmo SA, puede mejorar el workflow makespan medio

hasta un 34%, el uso de recursos hasta un 86%, y la eficiencia media de los recursos hasta un 96%, en comparación con las mismas métricas obtenidas con FCFS de SLURM en clústers. En la segunda ronda de experimentos de validación hemos demostrado que el HBRM con el algoritmo Biobackfill puede mejorar el makespan medio de los workflows en verso Firstfit y Bestfit un 8.55 %. En la tercera ronda de experimentos de validación demostramos que el HBRM con el algoritmo SA puede mejorar los makespans de epigenómico, Montage y Sipht multiworkflows obtenidos con Min-Min y Max-Min en clústers con un gran número de nodos un 43%. Además, hemos demostrado que en clústers con un gran número de nodos y con las mismas cargas de trabajo, que el algoritmo SAFP puede mejorar el multiworkflow makespan del algoritmo SA un 88%, y el multiworkflow makespan de Min-Min y Max-Min un 93%.

Chapter 1

Introduction

1.1 Bioinformatics data analysis

Computer science is an area of research that has experienced significant progress over the past decades. Increasingly sophisticated computing devices equipped with more processing power and storage capacities are continuously being rolled out by manufacturers at an overwhelming pace. Software and hardware improvements have allowed for newer programming paradigms to conduct complex, time-consuming tasks at greater speeds with reasonable costs.

Progress in computing technologies has benefited multiple science fields, which have been provided with enhanced tools to deploy newer experiments. One of this fields is life sciences, dedicated to the study of living organisms. Life sciences encompass many research areas, such as bioinformatics, which focuses among other things on the analysis of the genome. The genome contains the hereditary material of organisms, and is encoded in a molecule called DNA, mostly formed by four nucleotide base pairs A, T, G, and C. Study of the genome helps scientists gather knowledge of the entire set of genes of an organism, their interrelationships, and influence on the development of organisms. With this knowledge, scientists may determine the influence between certain genes and diseases or the response to organisms to drugs, helping develop customized medical treatments based on a person's genetic profile.

The genomic material of an organism can be obtained by sequencing biological tissue with the help of sequencing machines. Sequencing technologies experienced a major breakthrough in 2007 with the irruption of the so-called Next Generation Sequencing technologies, also known as high-throughput sequencing technologies. NGS massively parallelizes the sequencing process, allowing for billions of genome fragments or reads to be sequenced in one run and generating gigabases of information. NGS have drastically reduced both the cost and time of genome sequencing by 80% over the past ten years [3]. The size of genomic databases, such as EMBL (European Bioinformatics Institute), or GenBank (National Center for Biotechnology Information), is estimated to double every 18 months [2].

The ever-growing volumes of genomic data are usually studied with workflows, which follow

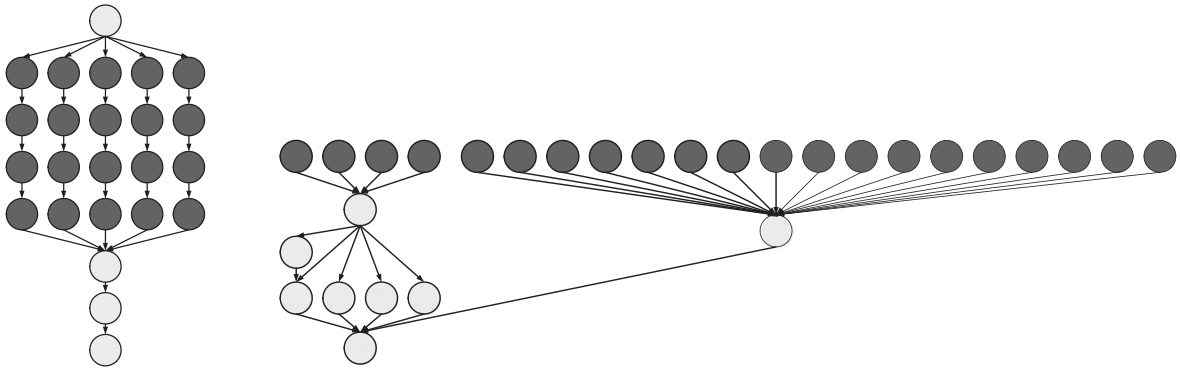
computational protocols to solve complex multi-step analyses. A workflow is a sequence of steps conducted by different computational tasks that are arranged in a predefined order with implicit dependencies. When workflow tasks are executed in computational nodes, they generate intermediate files. That is, output files which may in turn be the input of either a single posterior task (non-shared intermediate file) or multiple posterior tasks (shared intermediate file). The sizes of intermediate files may surpass that of the input files of the workflow. Once the posterior tasks receive all the intermediate files they require, their dependencies are considered solved and are ready to start execution. In many cases, workflows are modeled as Directed Acyclic Graphs (DAGs), where the edges represent the dependencies and the nodes of the graph represent the tasks. There exist different DAGs structures, the most common one is the simplified DAG, which unlike the extended DAG doesn't include parallel loops or conditionals [63].

Bioinformatics applications are the main tools used by data analysts or biologists to conduct the different workflow steps of the biological data analysis. There exist many kinds of bioinformatics applications, performing different tasks involved in genome analysis, such as: genome alignment, variant calling or annotation, among others. Genome alignment finds the location or position of a sample sequence in the whole reference genome by aligning the former to the latter. Variant calling searches differences between the aligned sample genome and the reference genome. Annotation is the process through which the differences or variants presented by the sample are reviewed, in order to determine their biological significance.

Users may integrate sets of bioinformatics applications to perform multi-step data analysis, forming bioinformatics workflows. They define the type of the analyses conducted in each workflow step (i.e. mapping, variant analysis..), and the order that will be followed. Additionally, they may select the bioinformatics tools that will execute the analysis, configure them to adjust the specifics of the analyses and achieve the desired behavior.

There exist many types of workflows, belonging to different research areas. Two popular workflows in bioinformatics are the Epigenomics workflow and the Sipht workflow. The Epigenomics workflow, represented in Figure 1.1 (a), performs sequencing genome operations. It splits the data into several chunks, operates on them in parallel, and merges the analyses results. The Sipht workflow, represented in Figure 1.1 (b), conducts wide searches for small regions of the untranslated RNA (obtained from the DNA) that regulate several processes such as secretion or virulence in bacteria.

Bioinformatics workflows are usually formed by complex pipelines with numerous tasks and datasets. Setting up those workflows for execution on large platforms may be time consuming and tedious. Users may have to pre-process the data to be analyzed, install the tools, and define their dependencies. Due to these complications, many users resort to Workflow Management Systems (WMS), softwares employed to build, customize, and run the sequence of computational or data manipulation steps of the workflows. WMS generate a representation of the whole computational procedure encompassed in the DAG-modeled workflow, and provide a visual interface so that



(a) Epigenomics workflow.

(b) Sipht workflow.

Figure 1.1 Examples of two well-known bioinformatics workflows

the user of the workflow can handle complex applications with little programming background. WMS are also useful for automating repetitive steps involved in the management of the workflows, and give an overall view of the structure and tasks. Galaxy [15] and Taverna [65] are two of the most popular WMS for building and executing bioinformatics workflows [32].

The ever-growing size of genomic databases and the need to analyze the information within, have boosted the popularity of bioinformatics applications. Shared cluster platforms have become common environments to execute bioinformatics applications. They provide users with access to powerful computing resources to fulfill the needs of the analyses conducted by bioinformatics applications. Many organizations research centers nowadays, dispose of data-analysis clusters to run their workflows.

However, disposing of powerful shared clusters doesn't automatically imply that the power is going to be correctly delivered to the different tasks submitted to the shared platforms to conduct data analyses, as this a challenging enterprise. Each task in the submission queue may have different resource consumptions, i.e. data-intensive or cpu-intensive, and therefore need different resources. Furthermore, all submissions are to share the cluster and will compete to use the different resources. Considering the different needs of the submitted workflow tasks when allocating resources for execution is essential for submitted workflows to achieve good performance, and minimize the time or cost of their executions. Furthermore, allocating cluster resources by considering the characteristics of queued tasks helps increase the amount of tasks simultaneously running in clusters, and increase resource utilization.

In this work, we address the problem of how to properly allocate tasks to shared resources, a NP-hard [43] problem long studied in research. We tackle this problem by considering a specific set of resources, workload, and performance metrics.

The computational resource considered to address this problem is a shared cluster formed by different multi-socket multi-core nodes with large memory capacities. The cluster is assumed to be equipped with a default Resource Management System, SLURM, which monitors the queue and includes a set of scheduling policies to allocate resources.

The workload considered to address this problem is formed by multiple bioinformatics workflow tasks which are submitted by different users to the shared cluster for execution. Each workflow has a series of different tasks, whose dependencies are assumed to be known by the system before submission. It is important specify that although we account for the task dependencies, we follow an approach that breaks the workflows down into a list of tasks, and considers for allocation those listed tasks that are in ready state.

Furthermore, we consider that users submit their workflows by specifying a performance criteria, which involves either the minimization of the workflow execution cost, or the minimization of the workflow makespan. That is, the time elapsed between the time instant the first workflow task enters the system to be executed, and the time instant the last workflow task terminates. Additionally, we include into consideration the criteria of the cluster administrator, which focuses on maximizing other performance metrics, such as the resource utilization or the efficiency.

1.2 Cluster Resources

This work addresses the problem of allocating a list of bioinformatics workflows to cluster resources. In this section, we review the approaches followed by current Resource Management Systems (RMS) on clusters to allocate workflow tasks to cluster resources.

The analyses conducted by bioinformatics applications are usually long-lasting and resource-demanding. On one hand, due to the storing and processing of large amounts of data files, which may contain up to millions of different sequences and weight several gigabytes. On the other hand, analyses are conducted by algorithms of outstanding complexity. The duration of the analyses depends on a large extent on the set of cluster resources allocated to the workflow tasks. In order to harness the computing power offered by clusters, and meet, in so far as possible, the user-defined performance criteria, such as workflow makespan or cost, it is essential to implement a proper resource managing approach on clusters.

Resource Management Systems on clusters are the entities in charge of handling the resources. Among the most common RMS on clusters one may find MAUI, Sun Grid Engine, or SLURM, which features in approximately 60% of the top 500 supercomputers. SLURM's popularity is due to its scalability and wide range of functionalities. It can operate on clusters with up to hundreds of thousands of nodes, and offers a wide range of functionalities brought by its numerous plug-ins. Furthermore, SLURM includes an interface for the development of plug-ins, slurm_spank, which facilitates users the process of integrating and adding their custom resource managing approaches. SLURM is composed by multiple daemons, such as slurmctld and slurmd. The slurmctld daemon runs on the management node and acts as a central entity that monitors system status. The slurmd daemons run on computing nodes and execute slurmd commands, such as the necessary ones to launch, kill, or monitor tasks [1]. Broadly speaking, the architecture of RMS is composed of a system monitor and a task scheduler. The system monitor monitors the

status of the submission queue (tasks, characteristics) and the status of the resources (workload, availability). The monitored information is sent to the task scheduler, which may contains multiple scheduling policies. Based on monitored information defining the cluster status, and the policy selected, the scheduler schedules the queue of tasks. On one hand, allocates a specific set of resources to each task in the queue for execution. On the other hand, decides with which priority each task in the queue is executed, and sorts the queue accordingly.

1.2.1 Scheduling Policies

Scheduling is a key topic of research in computer science. Many approaches have been developed over the years to schedule queues of tasks on cluster platforms. In this section we analyze taxonomy of the main scheduling approaches for scheduling both independent tasks and tasks within DAGs. The taxonomy is depicted in Figure 1.2, and followed throughout this section.

At the highest level, the scheduling approaches can be regarded as local or global. Local scheduling refers to assigning time slices of single-processor systems for concurrent execution. Global scheduling focuses on the assignment of tasks to multiple-processor systems such as clusters.

Global approaches can be considered static or dynamic. In dynamic scheduling, information involving DAG topology or the resources available is assumed to be unknown by submission time. In these cases, resource allocation is conducted on-the-fly as the application is running. Dynamic approaches are useful when it is impossible or unfeasible to estimate variables such as the execution time [29]. In static scheduling, information on cluster resources and tasks forming the workflows, such as precedences, is assumed to be somehow available by submission time [17]. Static scheduling approaches are the ones considered for this work.

Static algorithms can be divided in optimal and suboptimal. On cluster platforms, with many users attempting to access resources, scheduling decisions must be made fast. Scheduling shared resources to a queue of tasks is a NP-hard problem, as no algorithm is able to generate an optimal solution with polynomial time. In these cases, systems resort to suboptimal decisions which are made in an acceptable time. Suboptimal approaches can be divided into approximate, heuristics and meta-heuristics.

Meta-heuristics are problem-independent techniques generally applied to many different problems that don't have a problem-specific solution [10]. Examples of meta-heuristics are Genetic Algorithms or Simulated Annealing. Conversely, heuristics are defined in order to work out a specific problem. Scheduling algorithms generally follow heuristic approaches, as they provide good (not optimal) solutions within an acceptable time.

Scheduling heuristics can be divided into: list scheduling algorithms, individual task scheduling algorithms, clustering algorithms, and duplication-based algorithms. The simplest method is individual task scheduling, which makes decisions based on a single task. List scheduling algorithms are the most used ones, since they are relatively simple and less complex, compared

to other approaches [29]. The basic idea behind list scheduling algorithms is to assign priorities to the DAG tasks and place them in a list arranged in a decreasing order of priorities. Their operation mode consists of two phases: a task prioritizing phase, and a resource selection phase. Task duplication and clustering based scheduling algorithms focus on minimizing the communication delay between DAG tasks. An example is TANH [6], which replicates tasks to more than one resources in order to reduce the transmission time.

The purpose of the approach is to schedule workflow tasks that have been broken down into lists of tasks. Thus, we cast the focus on the list scheduling algorithms. Within the list scheduling algorithms, one may find the independency (batch) mode algorithms, the dependency mode algorithms, and the dependency-independency (hybrid) mode algorithms.

Dependency mode algorithms consider all the DAG-modeled workflow tasks, and set priorities (ranks) to each one of them based on: (i) their weight value, which is in turn calculated based on the execution time and communication time, and (ii) the rank value of their interdependent tasks (path time from a given ready task until the end). Popular examples of dependency mode algorithms are the Heterogeneous Earliest Finish Time First algorithm (HEFT) [57], which calculates the upward time of all tasks (times of all possible paths from each ready task until the end), and prioritizes the one with the largest upward time. Another relevant dependency-mode scheduling algorithm is ETF [26]. The main advantage of the dependency mode approach is the overall makespan reduction that they achieve by prioritizing those tasks with long interdependent-task times. However, their complexity is notorious since they must compute the whole critical path of all tasks. Namely, their complexity is $O(v^2,m)$, where v is the amount of tasks in the workflow and m the amount of processors [67].

In this work, we focus on scheduling a queue of bioinformatics tasks of different workflows. We assume that all task dependencies are known in advance. Although we account for task dependencies, we consider for allocation the ones that are in ready state.

Thus, we concentrated on batch-mode algorithms, whose complexity, $O(vgm)$ (where v is the amount of tasks in the workflow, m is the number of processors, and g is the number of tasks), is lower than that of the dependency mode algorithms.

Some of the most popular independency-mode algorithms are Min-Min and Max-Min [35]. The Min-Min algorithm calculates the overall completion time of all tasks and prioritizes the shortest one by assigning it to the node yielding minimum completion time. The Max-Min algorithm proceeds similarly to Min-Min, but sorting the tasks from maximum to minimum overall time.

There also exist classical scheduling policies to determine the order in which the tasks of the submission queue are executed on the cluster. The simplest one is First Come First Served (FCFS), which prompts the queued tasks to be executed in arrival order. Other approaches are Shortest Job First (SJF), which sorts the queue of submitted tasks from shortest to longest expected execution time, or LJF (Longest Job First), which sorts it from longest to shortest

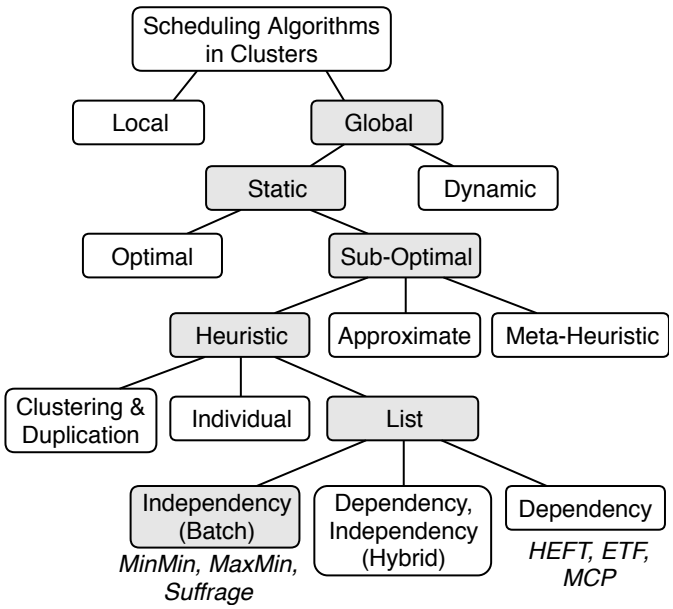


Figure 1.2 Taxonomy of the main approaches on clusters to schedule both independent tasks, and DAG-modeled workflow tasks.

expected execution time.

When these policies are implemented on cluster-submitted queues, scheduling gaps are usually generated, causing processors to remain idle over periods of time as other tasks wait for them to be released. To fill the scheduling gaps, the aforementioned policies can be combined with backfill techniques [22]. Backfill increases the initial priority of tasks (set by i.e. FCFC policy), in order to advance low-priority tasks in the queue and fit them on idle processors, as long as doing so doesn't delay the expected start time of any other task in the queue with higher priority.

Applying backfill enhances the cluster resource utilization obtained with classical policies, as resources that would otherwise remain idle are utilized by backfilled tasks. Furthermore, it reduces the waiting times of queued tasks, since they are executed before initially expected. Among the relevant backfill techniques in the literature, one may find Firstfit and Bestfit. Firstfit advances the first job of the queue that can be moved forward. That is, the first job that fits in a scheduling gap. Bestfit calculates the degree of fit of each job that fits in a scheduling gap, based on different backfill metrics: the number of processors, the execution time in seconds, or the product of both. Next, it advances the job with the best fit Ward et al. [60].

1.3 Performance of bioinformatics workflow applications on clusters

Bioinformatics workflow applications are significantly different than the majority of applications running on clusters. In this section we address the main particularities of bioinformatics applica-

tions by dividing them into two parts. First, the parallel paradigm in which they're programmed. Second, their variable performance.

One of the main differences between bioinformatics applications with respect to most others executed in such platforms is the parallel paradigm in which they're programmed. While most applications running on clusters are programmed in a distributed-memory paradigm, the majority of bioinformatics applications are programmed in a shared-memory paradigm [28, 56]. Their execution is conducted in a single node at a time by exploiting thread-level parallelism, as the datasets bioinformatics applications analyze can be easily divided into chunks and independently processed. In shared-memory architectures, threads share a unique memory space that is accessible for all node processors. Threads can easily communicate through the shared memory by means of variables or data structures, and the users don't have to incur in any programming efforts. However, shared-memory paradigms have their disadvantages too, since when increasing the number of threads, the number of paths between those threads and the shared memory is not increased. It is up to the programmer in those cases to define an appropriate strategy to access the shared data structures. For bioinformatics applications, usually branded as data-intensive, the memory bandwidth is an essential resource to achieve good performance, in those cases where a large number of execution threads are selected.

Conversely, distributed-memory paradigms divide execution into different processes which run on different-node processors. One of the main advantages of distributed-memory paradigms over the shared-memory ones is that they don't constrain execution to the amount of threads of a single node at a time, favoring scalability in a large number of different-node processors. As the multiple processes of an execution work towards a common goal, they must communicate. Nonetheless, communication between processes of different nodes is more complex than communication between same-node threads. The reason behind this is that each processor can only directly access the memory that belongs to its own node, and therefore, there is no shared memory address space to communicate through in an easy way (variables, data structures...). Distributed-memory inter-process communication is not only complex to program (must be done with message-passing or software interrupts), but is conducted through slower channels than in shared-memory approaches, and when large amounts of data are to be transferred through these channels, applications incur in overheads associated with the amount of time processes spend communicating. Many scheduling strategies have been developed focusing on minimizing the potential performance issues of communications among nodes, such as gang scheduling [11] or task clustering based techniques.

The second particularity of bioinformatics applications with respect to most executed on shared clusters is the variable performance. While the performance of most applications on clusters barely varies from one run to the next, the performance of a single bioinformatics application may be completely different in each run. Bioinformatics applications may adopt different resource usage usages, yield substantial makespan variation, and even show different

performance-bounding resources depending on the analysis case.

The highly variable performance of bioinformatics applications, adds an extra degree of uncertainty when attempting to allocate resources to bioinformatics.

The performance variation of bioinformatics applications broadly depends on two factors: the values given to the configuration parameters of each application, and the characteristics of the input dataset to be analyzed. Both the parameter values and the datasets are selected by the users of the applications before submitting them. The biological datasets of bioinformatics applications are encompassed in files, which are selected as input of the applications.

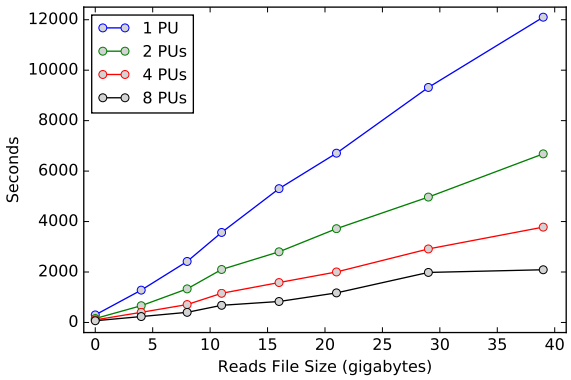
The characteristics of the datasets may have a great impact on applications' performance, such as the resource consumption or the makespan. Some of the major characteristics influencing the variable performance of bioinformatics applications are the number of sequences of the dataset, generally amounting up to millions, or the size of the file or files containing the datasets, which may reach several gigabytes.

Many different analyses can be run with bioinformatics applications on biological datasets. By setting the values of the different configuration parameters, users can adjust the specific kind of the analysis they wish to run. The chosen combination of parameter values dictates the complexity and duration of the analysis, and has a strong influence on the resulting performance of the application. Hence, parameter values must be considered when allocating resources.

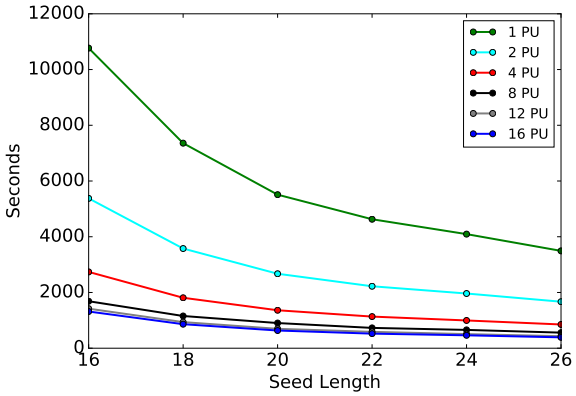
To illustrate the performance variation of bioinformatics applications upon parameters and data, we provide an example with bioinformatics read mapping applications. Mappers determine the exact position in the genome of the base pairs of different sequences (called reads sequences) of a partially-sequenced genome (stored in a file called reads file). To determine the exact position of each base pair (i.e. nucleotide), mappers compare the reads file with a fully-sequenced genome of the same species (reference genome), which is used as a template. Finally, the read mappers output a report with the different results or similarities (alignments) found when comparing the reads with the reference.

Depending on the kind of analysis specified by the values of the parameters, the application will output more or less alignments, each of which with an associated statistical significance. For instance, some analyses are intended for solely reporting high-quality similarities between reference and observed sample. These analysis call for deeper, more-through observations of the genomes than those others seeking lower-quality similarities. Finding high-quality similarities requires more exhaustive searches, and more complex algorithmic heuristics are applied. These analyses generally consume much more resources, and may prompt execution times to lengthen substantially.

Figure 1.3 shows the extent of makespan variation of two read mappers, Hisat and Bowtie, when modifying a single parameter or data characteristics. In Figure 1.3 (a), Hisat is executed with different reads file sizes, ranging between 1GB and 40GB, and mapped against the same human reference genome. As it can be seen, the makespan increases linearly with the size of the



(a) Makespan variation of Hisat with different reads sizes.



(b) Makespan variation of Bowtie with different seed length values.

Figure 1.3 Makespan variation of bioinformatics applications experienced when parameter values or data characteristics are modified.

reads file.

In Figure 1.3 (b), Bowtie is executed with identical input files, but different values of one of its parameters, the seed length. That is, the length of the subsequence within the target sequence. The seed length can be used to calibrate the commitment between sensitivity and speed. Shorter seed lengths yield high-quality alignments, but also yield large makespans. As the seed length increases, lower-quality alignments are found, resulting in shorter makespans. Increasing the seed length parameter a causes non-linear makespan reduction of around 65% regardless of the PUs used.

The performance variability driven by parameter values and datasets not only triggers substantially variable makespans, but also resource consumptions, usages and performance-boundedness. Current RMS approaches utilized nowadays on clusters to allocate resources to a queue of submitted tasks are not adapted to the particular performance of bioinformatics applications, and apply resource managing approaches oriented to a static resource consumption profile where applications always need the same amount of resources. Although some of the current RMS scheduling approaches resort to prediction to determine the resources applications need in future runs, predicting the resources that even a single bioinformatic application will need in the future is an arduous mission due to the many resource consumptions applications can adopt.

Current solutions on clusters are not aware of the particularities of the performance of bioinformatics applications, and don't dispose of enough knowledge as to approximate the resources applications will need. Thus, on clusters where a queue of bioinformatics applications of different workflows are submitted, the resources are allocated to each of the applications regardless of their needs. The lack of knowledge triggers inadequate resource allocation. Applications may be under-allocated resources, showing low performance and large makespans, or over-allocated

resources. In the latter case applications are allocated more resources than they actually need. Plenty of resources go low-utilized by applications, and as resources are already allocated (thus unaccessible by the rest), other applications pile up in the queue waiting to be granted access to resources. Resources go wasted, and the capacity of data analysis clusters to process the queue is compromised, dropping the performance of bioinformatics applications.

To adjust resource allocation to applications' needs, one may take into consideration the performance of the applications. One way to do so is to analyze information from past runs, and use it to estimate the resources that applications may need in the future.

1.3.1 Prediction Models

In this section, we review a taxonomy of the main prediction models in the literature to estimate the performance in tasks on platforms such as clusters. The taxonomy is depicted in Figure 1.4.

Many analyses in the literature have focused on predicting the execution time of jobs [52, 51], as well as their waiting time [44, 45]. Others studies have focused on predicting the slowdown time spawned when resources are shared [21].

At the highest level, prediction models can be divided in four categories: benchmarks, application code analysis, parameter prediction and simulation.

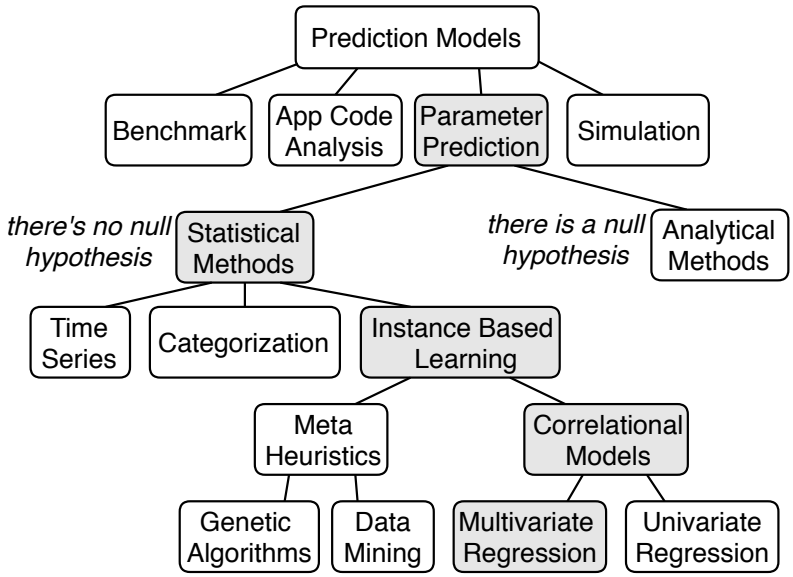


Figure 1.4 Taxonomy of the main prediction systems.

- *Benchmarks*

  Benchmarks are specific softwares that assess the and compare the performance of systems given a parameter of reference. They can be employed to evaluate software performance, e.g. the I/O activity of an application, or hardware performance, e.g. the CPU performance when

running an arithmetic operation.

- *Application Code Analysis*
  Application Code Analysis operate by inserting a series of checkpoints in the code of applications, to monitor and analyze it. They require thorough knowledge of the application code, which may become inefficient when dealing with multiple applications, due to cost and effort related issue. Application Code Analysis techniques allow analysts to gather knowledge on for instance, how much executing certain code chunks costs, or determining the amount of data transferred. However, they dismiss the system's characteristics.

- *Simulation*
  Simulation models artificially reproduce the behavior of a system in order to evaluate it. It is often used in cases where it is unfeasible to reproduce the real conditions in the system. I.e. because it is slow, costly or dangerous. Simulation can be used for instance to reproduce the performance of time-consuming applications on platforms such as clusters.

- *Parameter Prediction techniques*
  Parameter prediction techniques determine relationships between the coefficients and the variables included in a prediction function, in order to obtain an estimation of the desired metrics. Parameter prediction techniques may be adequate for determining the performance of bioinformatics applications, which as previously addressed, show a variable performance that must be captured when allocating resources. Parameter prediction techniques can be divided into analytical or statistical methods. Both of them are based on past experiences and employ statistics to conduct analyses.

- *Analytical Methods*
  Analytical methods usually deal with experiments with no assumed null hypothesis. In these scenarios, there is no model or formula that outputs the prediction values. They require more information than statistical methods, since researchers must build the model themselves. Analytical methods yield execution time predictions under specific system conditions. In other words, they only capture the static system features, dismissing the dynamic ones. Analytical models [49, 50] provide performance metrics like execution time, albeit they predict them by assuming the system, such as can be a cluster, has a determined, non-dynamic status.

- *Statistical Methods*
  Just as analytical models, statistical models are based on harnessing information from past

experiments. However, to generate predictions, statistical methods employ existing models. They yield good results when predicting the execution time of tasks, and can be divided into three categories: time-series, categorization and history-based learning.

- *Time series*
  Time series are discrete series of time-sorted data observations which are useful to foresee the availability of resources over a period of time, or predict tasks' performance upon load [64, 16]. They resort to the high time-correlation presented usually by systems' loads [27, 66], and can be suitable to model workloads and waiting times of shared systems. [18, 53].

  Both Categorization and Instance-based Learning (IBL) predict applications' performance based on past history. The difference between them resides on the nature of the historical information stored.

- *Categorization*
  Categorization techniques have been commonly used to predict tasks' execution times. To estimate the execution time of upcoming tasks, they use information from past runs that are stored in database [59]. In the database, categorization techniques include information about the task that has been divided into different categories. This approach doesn't include information of the status of the resources.

- *Instance-Based Learning (IBL)*
  Similarly to categorization, IBL uses historical database information to predict the execution time of applications. However, IBL not only includes information of the application but also of the status of the resources. When a new execution instance is received, IBL compares it with past instances conducted with similar resource status. IBL belongs to a branch of Machine Learning algorithms named Unsupervised Machine Learning (UML), where the computer is algorithm with labeled data by a supervisor or researcher.

  IBL approaches are suitable to determine the varying performance of bioinformatics, which depends such a great extent on the parameter values selected by users, as well as the characteristics of the dataset to be analyzed. IBL allows for the characteristics of the application to be categorized for future use, and considers the status of the resources. IBL methods can be divided into meta-heuristics, such as data mining techniques or genetic algorithms, and correlation models, such as univariate or multivariate regression models.

1.4 Resource allocation oriented to bioinformatics applications

This work focuses on the allocation of bioinformatics tasks to shared resources. In the past sections we approached current resource allocation approaches used by RMS to allocate resources to queues of tasks. We also reviewed the particularities of the performance of bioinformatics and explained why these approaches don't work out with bioinformatics applications, as well as their consequences: resource power waste and performance drop.

To prevent these issues, we introduce a novel approach to allocate bioinformatics applications to shared cluster resources. Given a set of cluster resources and a series of workflow tasks submitted for execution, the problem faced in this work is to determine how the shared cluster resources must be distributed across the different tasks of the submission queue, in order to maximize in so far as possible the criteria established by each user, the minimization of the each workflow makespan or cost, while also considering the performance criteria of the cluster administrator (maximizing resource utilization).

Most clusters are shared by many tasks running simultaneously, and their performance is also affected by a dynamic workload, which may vary over time. Applications no longer have full-node resources available, which generally ensured fast execution. Instead, they will likely be allocated logical processing units of a node, referred to in this work as processing units or PUs, i.e. a node containing 8 cores with hyper-threading accounts for 16 PUs.

On clusters with shared node resources, the amount of applications simultaneously running on each node, or Degree of Multiprogramming (DP), will grow bigger than one. Sharing node resources improves significantly the efficiency and resource usage the tasks. Furthermore, it reduces notoriously the waiting time of tasks in the queue, as multiple applications can be simultaneously in the same node instead of a single one. As a result, the overall performance. Despite the resource competition spawned, sharing resources improves the overall performance of tasks.

When users of bioinformatics workflows submit their jobs to clusters, they face the challenge of determining which combination resources is more suitable for application performance (makespan or cost constraints), given the parameter values, dataset characteristics, and time-varying resource availability. Finding the proper combination of resources for each case is a hard task, which is also crucial, as the resulting application's performance largely depends on it. The main goal of this work is to maximize the performance of bioinformatics workflow applications running on clusters with heterogeneous nodes.

In this work, we propose a new resource-managing approach that adapts the resource allocation of RMS on clusters to the particularities of bioinformatics applications.

Thus, we developed a History-Based Resource Manager (HBRM) for bioinformatics workflow applications running on clusters. The proposed manager, described in Chapter 2, can be

applied in environments hosting many kinds of applications with similar resource requirements. However, for the present study we have chosen to focus on the well-known context of bioinformatics applications with large datasets volumes, which represents a compelling example of a real data-processing domain. To build the HBRM, we followed a series of steps to identify the particular performance requirements of applications given specific combinations of parameters and datasets.

Given parameters and data of applications, the predictor generates multiple performance predictions, with different resources. Third, we developed scheduling algorithm which is fed with performance predictions and slowdown information. The algorithm determines how to allocate resources maximizing average workflow makespan or cost, efficiency and resource usage. Moreover, the algorithm considers the DP of the nodes, determining which combinations of applications make best-possible candidates for same-node execution, so that slowdown is minimized.

Based on conclusions from the taxonomy of the prediction models, we chose a prediction approach within IBL. Namely, we opted for a multivariate regression prediction model.

1.5 Workflow Simulators

As explained in this chapter, we address the resource allocation problem of multiworkflow tasks on shared clusters and propose a solution encompassed in the HBRM.

The performance yielded by queues of workflow tasks submitted to large clusters depends on a large extent on the resource managing approach followed to process it. Resource managing approaches can be tested by processing different queues of workflow tasks in different clusters. However, real-like environment testing (large clusters with heavily loaded conditions) becomes unfeasible, as the time to process the load may be substantially long, and acquiring large platforms with powerful resources carries on large costs.

Workflow simulators are suitable tools for evaluating and testing the performance of scheduling policies. They allow researchers to reproduce the referred working conditions by also saving costs and time. In this work we will resort to simulation to test the HBRM. To compare the current workflow simulators, and determine the most appropriate one to conduct our mission, we reviewed some of the most relevant workflow simulators.

- *SimGrid*
  SimGrid [13] is a generic framework to simulate distributed applications in Grids. It is an event-based simulator that provides a set of customizable tools to build the applications and configure infrastructures characteristics. SimGrid was developed to evaluate the performance of real large scale distributed environments, and model the resources by their latency and service rate [36].

- *GridSim*
  GridSim [12] is a discrete event simulator, with similar motivations than those of SimGrid. One of the differences its that it focuses on Grid economy when scheduling applications. The scheduling involves the notions of producers or resource owners, consumers, and brokers performing resource discovery and allocation. GridSim works on a higher-level compared to SimGrid, and is optimized to spot interferences between distributed brokers' decisions [46].

- *GangSim*
  GangSim [19] is a Grid simulation toolkit that provides support for modeling of Grid-based virtual organizations and multi-site resources. However, as many other simulators, lacks support for virtualization and heterogeneity when simulating the execution of applications. [36].

- *MAST*
  The MAterials Simulation Toolkit (MAST) [37] is an automated high-throughput workflow manager, and has two main operation lines: creating a highly-customizable workflow using an input file, and managing workflows. For the latter, it collects queue status information and information from the workflow directories. MAST uses this information to check the progress of individual jobs in the workflow, set up the next jobs when parent jobs complete, and check the overall completion status of the workflow Tam Mayeshiba [54].

- *tGSF*
  tGSF Hirales-Carbajal et al. [25] is a trace-based simulator employed to study Grid resource management problems. It contains mechanisms for parallel job interchange between sites in Grids. Site acceptance and job distribution are subject to policies, while scheduling strategies are configurable and provided by a so-called site layer [25].

- *CloudSim*
  CloudSim [48] is an event-based, cloud computing simulator built upon GridSim. It offers support for modeling, simulating, and instantiating many large-scale datacenters made up of multiple storage centers and physical host machines. One key asset of CloudSim is its support for virtual machines, created within the host resources [38]. Also, it includes support for user-defined policies. CloudSim can only deal with a single workload, but it is not suitable for workflow scheduling, as multiple tasks need to be scheduled together.

- *Workflowsim*
  Workflowsim [14] is an extension of CloudSim that includes a set of higher layers for managing and executing workflows with dependencies. Its higher layers, Workflow Mapper, Workflow Engine, Clustering Engine, and Workflow Scheduler, are explained in Section 3.1. Workflow scheduler is used to schedule the jobs to available resource according to the scheduling criteria defined by the user. While CloudSim conducts static scheduling during the planning

phase, i.e., jobs are scheduled statically before the execution of workflow starts, Workflowsim also supports dynamic scheduling. Jobs are scheduled to the remote scheduler when the corresponding resource becomes idle.

Workflowsim process workflows modeled as DAGs, and allows for scheduling algorithms to be validated in realistic scenarios. Workflowsim includes several algorithms such as Max-Min, Min-Min or FCFS. It allows users to easily add defined scheduling algorithms.

To deploy the experiments to validate the HBRM, we chose Workflowsim simulator, a popular simulation tool widely accepted within the scientific community. The criteria was made due to several reasons. Workflowsim considers the dependencies among workflow tasks, and disposes of a series of advantages to address the resource scheduling problem. On one hand, it allows for user-developed workflow scheduling algorithms to be included in its framework for testing. On the other, it features several state of the art scheduling algorithms, such as FCFS, Min-Min or Max-Min. Furthermore, Workflowsim is a flexible and editable tool, a paramount trait when it comes to modify some of the modules to fulfill the requirements of the user-developed algorithms. As for the resources, Workflowsim supports the creation of VMs, offering the possibility to configure them within hosts, and in turn, PUs within VMs.

1.6 Objectives and contributions

The main goal of this work is to improve the average makespan and resource usage of bioinformatics workflow applications on clusters achieved with current RMS. To do so, we developed a History-Based Resource Manager (HBRM) for bioinformatics applications from multiple workflows running on shared clusters with heterogeneous nodes. To accomplish this goal we defined a series of specific objectives which are listed below:

(1) *Design a methodology to characterize bioinformatics applications*. We defined a general methodology to characterize most bioinformatics applications. To test and expose the methodology, we defined a representative set of relevant applications commonly found in many bioinformatics workflows. The characterization methodology allows for relevant performance information gathering upon parameter values and data characteristics. Characterization information is subsequently stored in a historical database, and used to estimate the performance of bioinformatics in future runs.

(2) *Develop a Multivariate Regression Prediction Model*. The objective of the prediction model is to generate multiple makespan and cost predictions for each application submitted to the cluster, taking into consideration applications' parameter values and data characteristics. Multiple, different-resource predictions per submission allow for adjustment of

resource allocation to the actual needs of the task, regardless of resource availability. The predictor is to provide increasingly accurate predictions as more and more information is gathered in the database.

(3) *Minimize the makespan slowdowns of multiprogrammed nodes*. Increasing the DP of the nodes carries substantial overall performance benefits, which come along with makespan slowdown due to node resource competition. The goal of this specific objective is to minimize the makespan slowdown of applications in multiprogrammed nodes. To do so, we analyze in depth the resource usage of bioinformatics applications, and identify resource usage patterns, and potential performance bottlenecks. Based on resource usage information, we determine the combinations of queued tasks that make best candidates to be scheduled in the same nodes, so that the makespan slowdowns spawned in multiprogrammed nodes are minimized.

(4) *Develop a Multicriteria scheduler*. Based on applications' performance predictions and slowdown information, the goal of the Multicriteria Scheduler is to minimize either the makespan or the execution cost of the submitted workflows, while also maximizing the resource utilization. To accomplish this goal, three scheduling algorithms were developed:

   (a) *Slowdown-Aware (SA) Scheduling Algorithm* : adjusts resource allocation of each task in a cluster submission queue to their actual resource needs (application, parameters, data). Schedules combinations of queued tasks in such a way that the makespan slowdown of applications sharing multiprogrammed nodes (DP$>$1) is minimized.

   (b) *Biobackfill Scheduling Algorithm*: modification of the SA algorithm that applies backfill.

   (c) *Slowdown-Aware File-Placement (SAFP) Scheduling Algorithm* : extension of the SA algorithm which in addition considers all the different cluster memory hierarchy levels (shared NFS unit, nodes' local hard disks and nodes' local RAM disks) to place requested workflows files (based on their size and whether they're shared).

(5) *Validate the HBRM with the SA algorithm on clusters with multiworkflows*. Validate the developed HBRM (formed by the following blocks: Application characterization, Multivariate Regression Prediction Model, and Scheduler Multicriteria) with the SA scheduling algorithm on clusters where multiworkflows are submitted. As the cluster includes a RMS by default (SLURM), the HBRM doesn't operate by itself. It is also an objective to prove that the HBRM is capable of operating alongside SLURM. To validate this point we prove

that on shared clusters where a queue of multiple bioinformatics workflow applications is submitted, the HBRM with the SA algorithm improves the average workflow makespan, resource utilization and efficiency of SLURM's FCFS.

(6) *Validate the HBRM with the Biobackfill algorithm on clusters with multiworkflows*. Validate the developed HBRM with the Biobackfill scheduling algorithm on clusters with multiworkflows using Workflowsim simulator. To validate this point we prove that on shared clusters where a queue of multiple bioinformatics workflow applications is submitted, the HBRM with the Biobackfill algorithm improves the average workflow makespan obtained with Bestfit backfill and Firstfit backfill.

(7) *Validate the HBRM with the SA and SAFP algorithms on large clusters with multiworkflows*. Validate the HBRM with the SA and SAFP algorithms on large clusters with multiworkflows, using Workflowsim. To validate this point we prove that on shared clusters where a queue of multiple bioinformatics workflow applications is submitted, the HBRM with the SA and SAFP algorithms improves the average multiworkflow makespan obtained with Min-Min and Max-Min scheduling algorithms.

The main contributions generated in this thesis are the following ones:

- Generation of a History-Based Resource Manager for bioinformatics applications on shared clusters, composed of three main blocks: Application characterization, Multivariate Regression Prediction, and a Multicriteria Scheduler that includes the SA scheduling algorithm. When implemented on a queue of multiworkflows with bioinformatics applications submitted to shared clusters, the HBRM with the SA scheduling algorithm improves the average workflow makespan by up to 34%, the average resource utilization by 86%, and the average resource efficiency by 96%, compared with SLURM's FCFS. This contribution and results were published in an article titled: A Resource Manager for Maximizing the Performance of Bioinformatics Workflows in Shared Clusters, presented in the 5^{th} International Workshop on Parallelism in Bioinformatics (PBio) of the 17^{th} International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP-2017), in June 2017. This paper was qualified among the Best Papers of the conference.

- Implementation on Workflowsim simulator of the HBRM with the SA algorithm. Prove that, when implemented on a queue of multiworkflows with bioinformatics applications submitted to shared clusters, the HBRM with the SA algorithm improves the average workflow makespan obtained with Min-Min and Max-Min scheduling algorithms. The results were published

in an article titled: A history-based Resource Manager for Genome Analysis Workflows Applications on clusters with heterogeneous nodes, published in the International Journal on Parallel Programming (IJPP), in September 2018.

- Development of a new scheduling policy for the History-Based Resource manager for bioinformatics applications on shared clusters, named Biobackfill scheduling policy. When implemented on a queue of multiworkflows with bioinformatics applications submitted to shared clusters, the HBRM with the Biobackfill scheduling policy improves the average workflow makespan obtained with Bestfit and Firstfit algorithms by 8.55%. This contribution and results were published in an article titled: Bio-backfill: a scheduling policy enhancing the performance of bioinformatics workflows in shared clusters, presented in the International Conference on Complexity, Future Information Systems and Risk (COMPLEXIS), in April 2018.

1.7 Structure of the document

The present document is broadly divided in five main chapters.

(1) In Chapter 1, we describe the problem addressed in this work, the resource management on clusters hosting bioinformatics applications from multiple workflows. We start emphasizing the increasingly-important role played by bioinformatics applications, and the ever-growing volumes of data to be analyzed by them in platforms such as clusters. We address the complexity of the general, NP-complete resource management problem, and expose the particularities shown by bioinformatics applications compared to most other applications traditionally executed on clusters. We study the approaches taken by current RMS on clusters, and analyze the main scheduling policies they applied, as well as the main prediction mechanisms from which those scheduling policies can extract information to operate. Next, we address the reasons why current approaches followed by RMS don't work well when scheduling a queue of bioinformatics workflow tasks, review the consequences, and propose a new solution. The proposed solution is a new History-Based Resource Manager (HBRM) for bioinformatics workflow applications on shared clusters. Finally, we provide a taxonomy of the main frameworks to simulate the execution of scientific multiworkflows on clusters.

(2) In Chapter 2, we introduce the core proposal of this work, the History-Based Resource Management (HBRM) for bioinformatics workflow applications running on shared clusters. The HBRM is composed of three main blocks which we cover in depth: Application Characterization, Multivariate Regression Predictor, and Scheduler Multicriteria.

(3) In Chapter 3, we explain the modifications done on the Workflowsim simulator to enable the implementation of the SA scheduling algorithm included in the Scheduler Multicriteria of the HBRM. Adapting Workflowsim to the needs of the HBRM algorithms, will allow us to validate the HBRM in large clusters.

(4) In Chapter 4, we introduce the diverse sets of experiments to validate the HBRM with all the scheduling algorithms: the Slowdown-Aware (SA) scheduling algorithm, the Biobackfill scheduling algorithm, and the Slowdown-Aware File-Placement (SAFP) scheduling algorithm. Validation is done by processing different multiworkflows in different clusters with the HBRM algorithms, as well as with state of the art policies. Doing so allows us to comparing different performance metrics such as makespans, resource efficiency and resource utilization obtained with the different approaches.

- In Section 4.2, we validate the HBRM with the SA algorithm on clusters, and prove that it can improve the average workflow makespan by up to 34%, the average resource utilization by 86%, and the average resource efficiency by 96% obtained with SLURM's FCFS.
- In Section 4.3, we validate the modifications done on Workflowsim simulator to be able to implement the SA algorithm, and prove that only a 8% average workflow variation is introduced when modifying the Workflowsim simulator to implement the SA algorithm of the HBRM. Also, we validate the HBRM with the SA algorithm on large clusters by proving it improved the average workflow makespan obtained with Min-Min and Max-Min.
- In Section 4.4, we validate the HBRM with the Biobackfill algorithm on clusters, and prove that it can improve the average workflow makespan obtained with Bestfit and Firstfit algorithms by 8.55%.
- In Section 4.5, we validate the HBRM with the SA and SAFP algorithms on large clusters. We prove that the HBRM with the SA algorithm can improve the makespans of Epigenomics, Montage, and Sipht multiworkflows obtained with Min-Min and Max-Min by on large clusters 43%. Also, we prove on large clusters and with the same workloads that the SAFP algorithm can improve the multiworkflow makespan of the SA algorithm by 88%, and the multiworkflow makespan of Min-Min and Max-Min by 93%. This line of work and results are recent and considered for future publication.

(5) In Chapter 5, we summarize the benefits attained by the HBRM and expose the conclusions extracted from this work, as well as drawing the major lines encompassing future steps that can be taken to enhance its improvements.

Chapter 2

A History-Based Resource Manager for bioinformatics workflow applications

In this section we address the issue of how to properly allocate shared cluster resources to a queue of bioinformatics workflow applications for execution, and introduce a novel solution to the problem.

We consider a scenario in which users submit multiple workflows with different bioinformatics applications to a shared cluster for execution. Each application has different resource requirements, and users submit them by specifying the performance criteria they want to optimize: either the makespan or the cost of their workflow execution. Furthermore, we consider the criteria of the cluster's administrator, maximizing the utilization of the cluster resources.

The cluster considered is formed by various multi-socket multi-core nodes with diverse architectures, memory capacities, and amount of processing units (PUs). As the cluster is shared, multiple applications from different workflows are executed on the cluster nodes, and compete for the same resources.

Another consideration is that clusters come along with their own default RMS, which in turn include a set of scheduling policies. The RMS determine the priorities and resources of queued tasks, such as the amount of PUs that they are executed with, or the cluster node that will host the execution. Resource allocation is a topic of uttermost importance in these scenarios as the quality of the performance metrics obtained after queued workflows are processed largely depends on it.

RMS on clusters allocate resources among queued tasks accounting for the characteristics of the majority of applications on clusters, such as the distributed-memory applications employing MPI libraries. They allocate resources by focusing on the major resource needs of these applications, and are oriented towards minimizing the consequences of potential bottlenecks. One example is HEFT, which sets priorities to each tasks based on their execution time and communication time, which is significant and must be considered for distributed-memory applications communicating different-node processes, but is much less significant [57] in shared-memory applications such as bioinformatics, which communicate same-node processors quickly

through the shared memory address space.

Furthermore, the resource usage and consumption of the traditional applications on clusters barely changes from one execution to the next, and RMS consider this fact to, based on past runs, determine the resource needs of tasks in next runs.

Nonetheless, this approach should not be applied with bioinformatics applications, whose resource usage, consumption and even resource-bounding resource can change from one run to the next. Variation is due to the characteristics of the input datasets to analyze (file size, number of sequences...), and the parameter values, which determine the depth and complexity of the analysis run with the application on the dataset.

As a result of overlooking the major particularities of bioinformatics applications, the amount of resources allocated by traditional approaches of RMS on clusters doesn't adjust to the actual resource needs of such applications, compromising their performance and wasting cluster resources.

The goal of this work is to improve the way current RMS allocate bioinformatics workflow tasks to shared cluster resources in order to improve the overall makespan or cost of the workflows, as well as the resource utilization and efficiency. To address this problem, we focus on the performance-driving characteristics of the applications within each workflow submitted. In this work we assume that all workflow task dependencies are known in advance. Given a submitted queue of workflow tasks, our approach starts by breaking down all workflow tasks and creating a resulting of tasks. This list will contain both ready and non-ready tasks. In each scheduling iteration our approach considers for allocation those tasks in the list that are in ready state.

Given a queue of multiple bioinformatics workflow tasks with different resource needs (applications, parameters, and data), which is broken into a list of ready tasks and submitted to a shared cluster with variable workload and multiprogrammed nodes (DP$>$1), we propose a novel resource managing approach that determines how allocate the queue of tasks to shared cluster resources in such a way that once processed the queue, the overall workflow makespan, efficiency, and cluster resource utilization are maximized.

The novel resource managing approach is a History-Based Resource Manager (HBRM) for bioinformatics workflow applications running on shared clusters. The HBRM takes a queue of multiple workflows of bioinformatics applications submitted to a multi-core multi-socket queue clusters, breaks the multiworkflow tasks into a list of ready tasks, determines the amount of workflows admitted for execution on the cluster, and based on the applications, parameters and data of each task, proceeds as follows:

- Predicts the resources that each application needs, based on performance information from past runs.
- Schedules different applications for same-node execution, based on a compatibility table that accounts for the DP of the nodes

When users submit bioinformatics workflow tasks to a shared cluster, given the application, the parameter values, data characteristics and resource availability, the HBRM determines the adequate combination of resources to attain in so far as possible: (i) users' selected criteria, makespan or cost minimization, and (ii) the administrators' criteria: maximizing the resource usage.

Furthermore, as doing so, the HBRM automatically estimates the amount of resources of the bioinformatics applications, and adjusts the real resource demands of the multiworkflow to the existing resource capacity held by the cluster. In Figure 2.1, we display the HBRM with its main blocks in red color, implemented on a cluster.

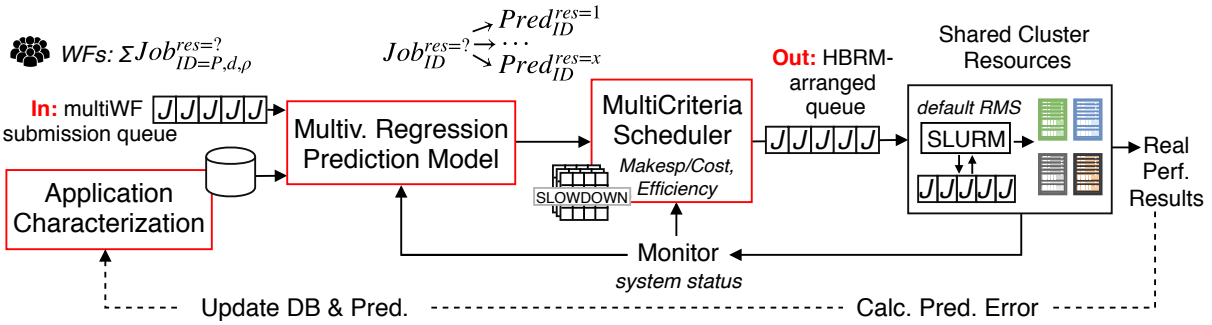


Figure 2.1 History-Based Resource Manager (HBRM), with its main blocs in red color, implemented on a shared cluster.

We also considered, as shown in Figure 2.1, that clusters or datacenters analyzing the considered biological data where the HBRM can be implemented already include a RMS by default, e.g. SLURM, which allocates applications to resources as dictated by the policy chosen. To improve resource allocation of RMS while also considering that cluster have their own default-RM policies, the HBRM, placed inside the cluster, receives the submission queue before the default RMS does. The HBRM allocates resources and sets priorities to the submission queue by considering the characteristics of bioinformatics workflow applications, and as can be seen in Figure 2.1, generates an HBRM-arranged queue which is sent to SLURM. With this strategy, SLURM receives a queue with implicit information on the amount of resources each task actually needs, such as the exact amount of PUs, as well as which combinations of queued tasks make best candidates to be simultaneously executed in the same node. Later on, SLURM, based on its implemented policies, and the implicit information of the HBRM-arranged queue, schedules the queue.

The main blocks of the HBRM, Application characterization, Multivariate Prediction Regression Model, and Multicriteria Scheduler, are described in this section.

- Application Characterization block, described in Section 2.1. We introduce a general methodology to identify and classify the main performance-driving characteristics of bioinformatics applications. We design a series of characterization experiments in which multiple bioinfor-

matics applications are executed on the cluster with different parameters, data, and resources. We create a performance monitor and a historical database to store the performance of each run conducted on the cluster. The goal of this block is to generate a continuously-growing amount of historical application performance information of past all runs conducted on the cluster, to be able to predict the resources applications will need in future runs.

- Multivariate Regression Prediction Model, described in Section 2.2. Users can submit their bioinformatics applications selecting one out of many possible combinations of parameters and data values. However, the workload of clusters hosting bioinformatics workflows shows minor variations over time, as most bioinformatics workflows follow established protocols, and formed by common applications, parameters and data. This trait is taken advantage of by the predictor. Based on performance information from similar past runs (application, parameters and data) conducted on the cluster and stored in the database, the predictor estimates the resources queued tasks need in future runs.

- Scheduler Multicriteria, described in Section 2.3. We analyze the predicted resource usage and consumptions of bioinformatics applications, to determine, given a series of cluster-submitted workflow tasks awaiting execution on multiprogrammed nodes, which combinations of queued applications make better candidates for same-node execution so that the slowdown is reduced to minimum. Based on prediction performance and slowdown information, we develop three algorithms: the Slowdown-Aware algorithm (Algorithm 1), the Biobackfill algorithm (Algorithm 2), and Slowdown-Aware File-Placement Algorithm (Algorithm 3).

  The SAFP algorithm has been developed in collaboration with previous work done on another research line [4], which focuses on maximizing the performance of bioinformatics workflows by accounting for the location of the workflows' input and intermediate files in the different levels of the cluster's memory hierarchy.

In the following sections, we review on by one the different blocks of the HBRM shown in Figure 2.1, starting by the Application Characterization block.

2.1 Application Characterization

Characterizing applications helps obtain information on how applications perform under different resources, parameters and data.

By characterizing applications, one may for instance observe applications' performance response when executed with different combinations of resources. This information is paramount for developing a good resource managing approach, as it may help determine the amount of resources applications may need. With enough characterization information, the HBRM can shorten other-RMS' gaps between the quantity of resources allocated and the quantity of

resources applications need to yield good performance. With performance information obtained by characterizing applications, the resources can be adjusted to applications' actual needs and overall performance maximized.

In bioinformatics it is common to find a recurring subset of applications conducting similar analyses in many different workflows. Thus, it is likely on cluster queues to find the same applications from multiple workflows used for regular data analysis sessions.

To raise knowledge on how applications behave, and spot their main performance-driving characteristics, we developed a method to characterize the performance of bioinformatics applications on clusters, applicable to most bioinformatics applications. It includes a series of experiments where performance is monitored and stored in a database to subsequently generate predictions.

The HBRM has been built by following a general approach considering most bioinformatics applications, so that it can be applied on clusters regardless of which bioinformatics applications are hosted. However, to start building the HBRM, and explain how it's been HBRM built, we selected set of bioinformatics applications. As the approach followed to build the HBRM is a general one, it can adapt to most bioinformatics applications other than the ones of the set.

To choose the applications of the set, we analyzed multiple workflows of different bioinformatics studies in the literature, such as: statistical methods for phylogeny estimation focusing maximum likelihood (ML) [34] or estimation of alignments and trees for single genes [61]. Other studies reviewed analyzed performance of different workflow tasks conducting whole-transcriptome [9] mapping, and mapping of DNA sequences obtained with different sequencing technologies [42]. Also have been considered for review articles analyzing the accuracy of mappers involving short reads [47], or aligners [7].

From these scientific contributions we extracted a summary of the most frequently-used applications from the published workflow applications. The resulting set of well-known applications is displayed in Table 2.1. The set is formed by different sequence aligners, mappers and phylogeny analyzers.

Table 2.1 Set of representative bioinformatics workflow applications chosen to explain the characterization methodology.

Aligner	Mappers						Phylogeny			
Blast	Bwa-mem	Bwa-aling	Bowtie	Hisat	Soap	Star	Phyml	Mrbayes	Fasttree	Raxml
2.6.0	0.7.5a	0.7.5a	2.2.6	2.0.5	2.21	2.4.2a	2.4.5par	3.1.2h	2.1.3.c	8.2.9

The first step to characterize bioinformatics applications consisted in determining the numerous and configurable factors conditioning their variable performance when executed on clusters. These factors were classified in four categories: configuration parameters ($P = P_{1,..,x}$), input datasets characteristics or data ($d = d_{1,..,y}$), application user's performance criteria (ρ: makespan, cost), and the resources used (*res*). From this classification we generated a vector representing the execution of the applications on clusters, displayed in Equation 2.1. The execution vector

allows us to capture and classify the different behaviors shown by applications upon given parameters and data.

$$Job_{ID} = App^{res}_{P,d,\rho} = App^{node,PU,mem}_{P_{1,..,x},d_{1,..,y},\rho} \tag{2.1}$$

In the following sections we cover the factors conditioning the performance of bioinformatics applications: parameters, data characteristics and resources. Based on this characterization, the performance experiments will be carried out.

2.1.1 Configuration parameters

The configuration parameters of bioinformatics applications determine the kind of analysis to be run on input datasets. By giving different values to parameters, users can for instance adjust the quality or quantity of the results obtained with genome mappers or phylogeny tools, such as those of Table 2.1. The parameter values of bioinformatics applications must be considered, as depending on them, applications can yield substantially different performance: makespans, resource usage or consumption.

Bioinformatics applications may have many configuration parameters, most of which can take a wide range of possible values to choose from. The number of parameters and ranges of values result in a huge amount of possible parameter value combinations that applications can be executed with. Blast aligner for instance, has 47 different configuration parameters. Assuming each can take a range of, say 4 values, would generate 5 million possible combinations. The amount of possible combinations grows exponentially with the range of values. For instance, a 5-value range per parameter would result in 230 million possible combinations for Blast to be executed with when analyzing a single dataset.

The number of possible parameter value combinations form an enormous parametric space that is unfeasible to analyze in its entirety. In practice however, we found out that common bioinformatics applications are usually executed by focusing on a subset of parameter values, fact that we contrasted in the literature [31, 33, 30]. Hence, we started by focusing on these parameters. Nonetheless, less-frequent submissions including containing uncommon parameter values are also taken into account by the HBRM, as the performance information of all executions on the cluster (including those with rarely-selected parameter values) is monitored and stored in a historical database.

In two different lists below we review some of the performance-determining parameters used (observed and given different values) in many analyses conducted with the bioinformatics aligners, mappers, or phylogeny tools of Table 2.1. Although their names or functions may not be identical from one application to another, they are somehow equivalent, with slight changes. The first of the two lists includes parameters of aligners and mappers, which compare two sets of sequences contained in different files. The parameters below strongly affect the comparison

procedure and resulting application performance:

- *Word size, seed length*
  Aligners and mappers commonly start by breaking each read sequence into chunks called words or seeds, and initially align them to the reference genome. The word size or seed length is the amount of base pairs of each word, and in turn, the whole set of words generated from each sequence is called word list. The word size parameter can be used to adjust the sensitivity and speed of sequence searches. Increasing the word size may result in a shorter word list, speeding up the word-reference sequence similarity searching process. Nonetheless, it may also cause the search to miss some alignments of shorter lengths. Conversely, shortening the word size may yield more alignments, but intensifies the complexity and lengthens the runtime of the analysis.

- *Threshold score to start extending alignments*
  Aligners and mappers attempt to *initially* align or map words to the genome. Matches are not always identical but similar. The degree of similarity between a word and a query is measured with the score parameter. The word-to-genome comparisons yielding a score above a certain threshold, are kept and considered initial alignments. Later on, they are extended to generate *final* alignments. The threshold score for extending alignments can also be used to adjust the search quality and depth. The higher the threshold, the better quality initial alignments are pursued, and the more potentially-initial alignments are discarded. This speeds ups the search, but can cause the application to overlook not-so-good initial alignments which could have been relevant once extended.

- *Cut-off value, or score threshold to stop extending alignments*
  The words producing initial alignments are saved, and used as seeds. They are extended in both directions in order to find final alignments, a process known as seed and extend. As seeds are extended both left and right, the new score is calculated again. The extension is carried out until the score drops below a second threshold, the cut-off value. The lower the cut-off, the more comparison steps are conducted. Thus increasing complexity and time, yielding lower-quality though numerous results. The higher the cut-off, the higher the quality of the results, albeit lesser results are obtained.

- *Hits*
  The hits value is calculated with the Karlin-Altschul equation [5], and determines how often an alignment with a given score is expected to occur by chance. That is, its statistical significance. A default hits value of 10 implies that 10 matches are expected to be found by chance. If the statistical significance of a match is greater than the expected threshold, the match isn't reported. Low expected values generate lesser chance matches of high quality, increasing the makespan of the application. Conversely, high expected values generate many chance matches of low quality, which results in a faster execution.

- *Gap open penalty and mismatch penalty*
  Due to errors, sequenced reads may have insertions (added bases) and deletions (missing bases). Gapped alignments take into account both insertions and deletions. Ungapped alignments are faster than gapped alignments, although less sensible. Gaps usually have a penalty associated, just as mismatches.

The second of the two lists includes parameters of phylogeny applications which are included in the initial set of applications of Table 2.1.

Phylogeny refers to the study of the evolutionary history of groups of species, and their relationship with other groups with common ancestry. As a result of the analyses, a phylogeny tree is built. The tips of the tree represent the descendant or species, whereas the nodes represent the common ancestors of those descendants. Phylogeny trees allow for instance, to determine the number of necessary divisions to obtain a cell, and analyze mutations generated through the division process.

- *Substitution model*
  Probabilistic substitution models are used to estimate the tree that best fits the aligned sequences, and define how each species is related to one another. The most common way to determine the tree topology is with Maximum Likelihood (ML). The substitution model can have a notorious impact on the resulting makespan of phylogeny applications.

- *Bootstrap number*
  Bootstraping, proposed by Felsenstein in 1985 [20], calculates the confidence of the estimated phylogeny trees. It assesses the confidence of clads, that is, the branches of the phylogeny trees depicting a species and its ancestors. Bootstrap analysis starts by re-sampling sequence data with replacement, forming so-called bootstrap samples of a size identical to that of the original one. Next, generates pseudo-replicate trees on re-sampled data. Given a common ancestor (CA) with two descendants (S1, S2), a bootstrap value of 90% would imply that S1 and S2 would be siblings in 90% of the bootstrap replications. The larger the bootstrap parameter, the more trees are generated, spawning greater makespans.

As explained before, bioinformatics applications have up to tens of parameters. Some of them are much more frequently used or given different values than others, and are commonly brought up in many bioinformatics experiments. To expose the resource managing approach, we focused on a subset of five parameters per application, included in Table 2.2.

Identifying and understanding the recurrent parameters of bioinformatics applications can help understand why bioinformatics applications performance varies from case to case, even when analyzing the same input dataset. Next, we approach performance variations triggered by datasets.

Table 2.2 Relevant parameters of the analyzed set of bioinformatics applications

App	Parameter 1	Parameter 2	Parameter 3	Parameter 4	Parameter 5
Blast	Hits	Word size	Mismatch penalty	Threshold for extending hits	Gap alignment drop off
Bwa-mem	Score	Re-seeding	Minimum seed length	Gap open penalty	Mismatch penalty
Bowtie	Number of mismatches	Seed length	Maximum mismatch penalty	Match bonus	Disallow gaps
Bwa-align	Max. num. of gap extensions	Maximum edit distance in seed	Take 1^{st} subsequence as seed	Gap open penalty	Mismatch penalty
Soap	Filter low quality reads	Totally allowed mismatches in 1 read	Match mode for read or seed	Report repeated hits	Allow gap size
Star	Filter mismatches in reads	Filter score < min (over read length)	Max. start length for seed search	Filter matches < min (over read length)	Seed
Hisat	Seed for random num. generator	Maximum mismatch penalty	Minimum mismatch penalty	Read open gap penalty	Read extend gap penalty
Phyml	Bootstrap number	Transition to transversion ratio	# substitution rate categories	Substitution model	Distribution parameter
Mrbayes	Structure of the substit. model	Nucleotide substitution rates	Number of runs or analyses	Number of generations	Sample frequency
Raxml	Parsimony random seed num.	2^{nd} substitution model	Random seed number	Substitution nucleotide model	Number of runs
Fasttree	Bootstrap number	Analyze multiple alignments	Number of rate categories	Reduce rounds of ML NNIs	Exhaustive search

2.1.2 Data characteristics

Bioinformatics applications analyze biological datasets stored in files, which must be selected as input of the given bioinformatics application. They are defined by a wide range of characteristics, such as the size, the number of sequences, or meta-data stating their associated quality.

Dataset characteristics are closely related to the resulting application performance, and must be considered when determining the amount of resources to be allocated. Some of the most notorious datasets characteristics are listed below:

- *Type of data*
  Biological data such as DNA is usually stored in text files containing strings of letters. Each string represents a DNA sequence, and is made up of for different repeated letters sorted in a specific order. Each letter stands for a different nucleotide base forming the DNA molecule: A (adenine), C (cytosine), G (guanine) and T (thymine). Biological data can also be represented as a set of aminoacids, the alphabet for proteins. Each three nucleotides codes for an aminoacid, resulting in 4^3=64 different aminoacids. For the experiments, we focused on nucleotide data. However, the HBRM has been built by following a general approach and it is equally valid for applications analyzing datasets of aminoacids.

- *Format of data*
  Biological input data can come in many formats. In this work we focus on the input data

file formats generally employed by mappers and aligners of Table 2.1, such as Fasta and Fastq. Fasta files contain sequences represented with two lines. One line includes the name of the sequence, and the other the string of aminoacids or proteins. Fastq files represent each sequence with four lines. The first one, contains the sequence name, and the second one contains the sequence of nucleotides or aminoacids. The third line includes a "+" character, and the fourth one includes a set of characters describing the quality of each base pair of the second line. That is, the likelihood that a base pair has been sequenced correctly.

- *File size, number of sequences, length of sequences*
  The file size is another relevant dataset characteristics that must be considered. When running analyses, algorithms may request data to be brought from memory or disk to the CPU storage units. The longer the file sizes, the greater the information flow that is conveyed back and forth. Plus, per each pair of reference-reads sequences, algorithms must generate a series of mapping steps. For instance, the Needleman-Wunsch [39] algorithm, generates a score matrix per each pair of analyzed sequences. The length of the sequences also influences performance. E.g., each scored matrix generated by the algorithm has a number of rows and columns matching the lengths of analyzed query-reference pair of sequences.

2.1.3 Resources

To generate the HBRM we used a cluster of four nodes with different characteristics: large memory capacities and different amount of logical cores, referred to as processing units (PUs). The main specifications of the cluster nodes are described in Table 2.3. The default RMS of the cluster is SLURM.

Table 2.3 Main specifications of the cluster partition used to characterize bioinformatics applications.

Node Architecture	RAM	PUs	Local Disk	Clock rate
AMD IO-6376	128GB DDR3	64	250 GB	2.3GHz
AMD IO-6376	128GB DDR3	64	250 GB	2.3GHz
Intel Xeon E5-4620	128GB DDR3	24	250 GB	2.2GHz
Intel Xeon E5-2620	64GB DDR3	24	250 GB	2.1GHz

2.1.4 Application Characterization Experiments

The idea behind the characterization of bioinformatics applications conducted in this work is to generate performance information of applications with different parameter values, datasets characteristics and cluster resources. This information is an initial volume of knowledge to asses the amount of resources such as PUs that future executions launched on the cluster will need, keeping in mind the aforementioned idea that bioinformatics data analysis sessions show minor variations over days or months, and are thus to resemble those made in the past.

When a new task submission (application, parameters, data) arrives in the cluster pending resources to be allocated, the HBRM, based on performance information obtained from past runs, estimates the resources the application will need. The more performance information previously obtained, and the more similar (parameters and data values) the past application executions are to the newly-submitted one, the greater the chances of the HBRM to accurately determine resources needed.

In the unlikely event a new task submission with notoriously different parameter values or data (from those seen in the past) is submitted to the cluster, the first time that happens, the HBRM won't be able to approximate the resources needed, and will follow a naive approach such as reservation of the whole set of PUs of a node. However, executions with notoriously different parameter values may come up on the cluster, and they won't be left out of the approach of the HBRM. Thus, as the new task finishes execution, the performance details will be monitored and saved in the historical database for later usage. The second time the task is submitted to the cluster, the HBRM will determine the resources of it based on the performance details stored the first time the execution was submitted, and so forth.

Once spotted the main parameters and datasets characteristics of bioinformatics applications, we designed a series of characterization experiments where applications were executed with different parameters, data and resources, to obtain performance information. These experiments enable us to tell how modifying each parameter and data characteristics affects performance.

To perform the experiments, each application of Table 2.1 was selected (for instance Blast), and each of their parameters of Table 2.2 was given a range of values (RangeVals):

$$Blast_{P1=RangeVals}, Blast_{P2=RangeVals}, Blast_{P3=RangeVals}, Blast_{P4=RangeVals}, Blast_{P5=RangeVals}$$

Next, we considered datasets of a range of sizes (d=Sz1, Sz2...), and different resources (all nodes with range of PUs). For instance, to determine the performance influence of a parameter (i.e. P1) of Blast application, with a given dataset (d=Sz1), we executed:

$$Blast_{P1=RangeVals;d=Sz1;res=(AllNodes),(RangePUs)}$$

Similarly, to determine the individual performance influence of Blast's P2, P3, P4 and P5, each with a given data (i.e. d=Sz1), we executed:

$$Blast_{P2=RangeVals;d=Sz1;res=(AllNodes),(RangePUs)}$$
$$Blast_{P3=RangeVals;d=Sz1;res=(AllNodes),(RangePUs)}$$
$$Blast_{P4=RangeVals;d=Sz1;res=(AllNodes),(RangePUs)}$$
$$Blast_{P5=RangeVals;d=Sz1;res=(AllNodes),(RangePUs)}$$

Once each application in the set is executed as explained for Blast, (different: range of

parameter values, dataset characteristics and resources), we can assess the performance influence of the parameters of each application with a given dataset regardless of the resources. To determine the influence of the dataset size on applications performance, the same approach follows. A range of sizes (RangeSizes) is selected, and the application is primarily executed with default parameter values:

$$Blast_{AllParams:defaultValues;d=RangeSizes;res=(AllNodes),(RangePUs)}$$

It may be given the case where a modifying the values of parameter (i.e. P1=RangeVals) doesn't apparently make a difference on resulting performance for a given case, lets say, when P2, P3, P4 and P5 are given default values. However, it may occur that the influence of P1 becomes noticeable only in the cases where P2 reaches certain values. To figure out the combined influence of multiple parameters and data, we executed:

$$AllAppsInTheSet_{(P1,P2,P3,P4,P5=RangeVals;d=RangeSizes;res=(AllNodes),(RangePUs)}$$

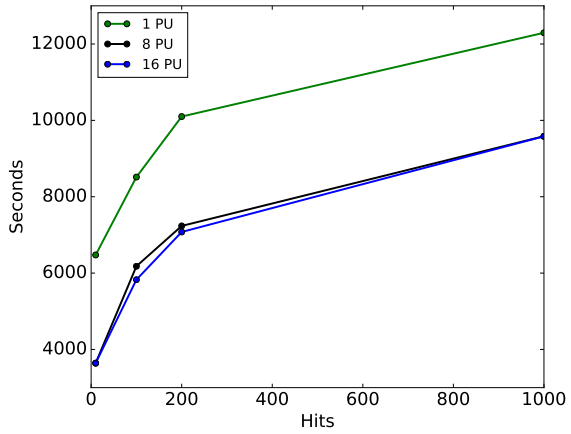
Some of the application characterization experiments done to determine the influence of one of the parameters of Table 2.2 on the resulting application performance, have been included in this section, and can be seen in the following figures. In each figure, we selected a node, and a dataset of certain size. We fixed all the parameters to default values, except the one whose influence on performance we wanted to analyze. To this parameter, we gave it a range of values. Figure 2.2 (a), and Figure 2.2 (b), show the influence on Blast application performance of the hits parameter value (Parameter 1), and the nucleotide mismatch penalty parameter value (Parameter 3), respectively. The makespan results of Figure 2.2 (a), were obtained by executing Blast application on the Intel Xeon E5-2620 node of the cluster of Table 2.3, with different hits parameter values and PUs.

The makespan results of Figure 2.2 (b), were obtained by executing Blast application on one of the AMD nodes of Table 2.3, with different nucleotide mismatch penalty parameter values and PUs.

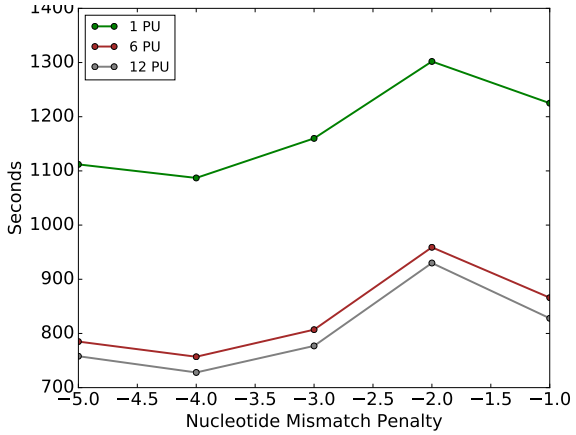
In both cases the Blast was executed with a database file of 4GB and 149 sequences of 1M base pairs of length, and a query file of 200kB and 308 sequences of 500 base pairs of length. Both figures display how the hits and the nucleotide mismatch parameter have a non linear influence on the resulting makespan of Blast.

Figure 2.2 (a), shows how the makespan, regardless of the amount of PUs, varies around 50%. Figure 2.2 (b), shows how the makespan varies non monotonically, interspersing increases and decreases as the number of PUs is increased.

Figure 2.3 (a), and Figure 2.3 (b), show the influence on Bwa-align application performance of the score parameter value (Parameter 1), and the minimum seed length parameter value (Parameter 3), respectively. The makespan results of were obtained in both cases by executing

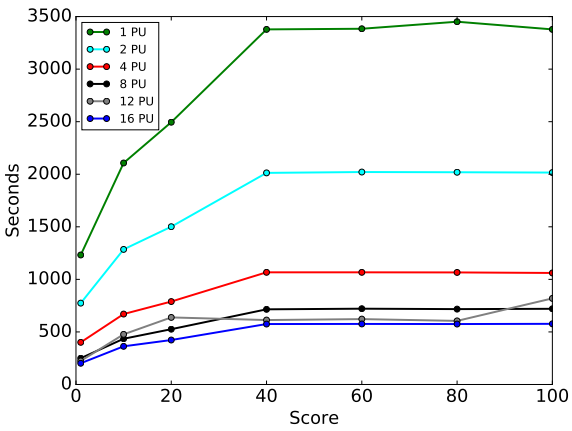


(a) Influence of the hits parameter value on the makespan of Blast .

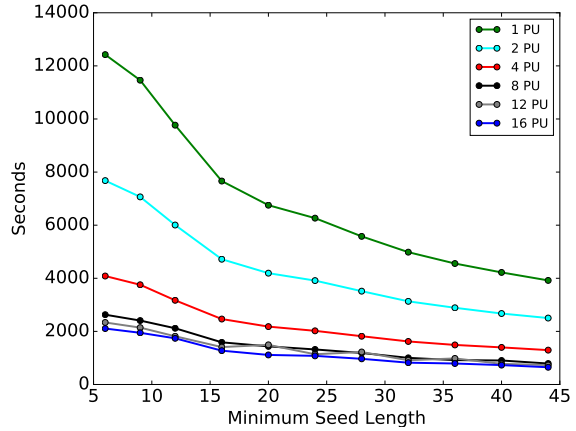


(b) Influence of the nucleotide mismatch penalty parameter value on the makespan of Blast.

Figure 2.2 Influence of two different parameters on the performance of Blast application.



(a) Influence of the score parameter value on the makespan of Bwa-align.



(b) Influence of the minimum seed length parameter values on the makespan of Bwa-align.

Figure 2.3 Influence of two different parameters on the performance of Bwa-align application.

Bwa-align on the Intel Xeon E5-4620 node of the cluster defined in Table 2.3, with different values of the respective parameters and PUs. In Figure 2.3 (a), we was used a reference file of: 5GB and 93 sequences of 34M base pairs of length, and a reads file of: 1GB of size and 4.5M sequences of 91 base pairs of length. In Figure 2.3 (b), it was used a reference file of: 5GB and 93 sequences of 34M base pairs of length, and a reads file of: 4GB of size and 18M sequences of 91 base pairs of length. Both figures display how the score and minimum seed length parameters have a non linear influence on the resulting makespan of Bwa-align. In Figure 2.3 (a), the makespan of Bwa-align increases around 100% as the score is augmented. In Figure 2.3 (b), increasing the minimum length of the seed causes a 200% makespan decrease with a single PU, and a 100% makespan increase with 16 PUs.

After performing all the experiments, a substantial amount of information is generated. This is an initial set of information that can be useful for the HBRM to know the resources

applications may need, as parameters and data chosen resemble those of most of the experiments usually conducted with bioinformatics applications on clusters. However this is not the whole information the HBRM will dispose of, since every time a new execution is submitted, new performance information is acquired. In the next section we detail how we gather, process, and structure this information in order to make the most of it in future runs.

2.1.4.1 Monitoring and Characterization Database

All the information generated in the application characterization experiments may be useful for future runs executed on the cluster, and represent the initial knowledge the HBRM comes with when first implemented in a cluster. However, this is not the only information the HBRM works with, as different executions may also be launched on clusters too, and when implemented, the HBRM must try to determine the resources for those cases too.

To do so, it is paramount for the HBRM to learn not only from information of the characterization phase, but also from all the experiments launched on the cluster on a daily basis. This way the HBRM will increase its performance information in an automated way, and know more about the resources applications need.

To harness information from each run launched by users on the cluster, we have set up a monitoring feature within the HBRM, represented in Figure 2.1. To monitor application performance of each run, a list of Linux monitoring tools are used, and exposed in Table 2.4: time, perf, sar, vmstat and pidstat. Metrics include: makespan, memory consumption and fails, cache loads, stores and misses at different levels, and amount of bytes read or written from disk. To minimize the overhead, the monitoring tools target the process ID of the application execution, leaving aside the processes generated by the monitoring tools themselves. Monitoring performance is saved in disk files, and later on consulted by the monitor to generate average values, standard deviations, median values or percentages.

Every time a cluster execution terminates, the monitored performance information is stored in a database. In this database it is also included the information generated in the application characterization experiments. Thus, the amount of information of the database grows by each execution launched on the cluster.

Every time a new job is submitted, the HBRM searches through the database for similar past executions that may have been executed on the cluster, in order to determine the resources the new submission may need. This process is depicted in Figure 2.4 with Bwa-mem application (BwaM) and with the makespan as performance criteria (ρ). The cluster receives a new submission: Job_{ID}=App=Bwa-mem$^{res}_{P,d,\rho=makesp}$. The HBRM then reads the values of the input attributes, represented in brown squares: application, parameters, data and performance criteria. Next, searches in the database for past executions done with: App=Bwa-mem, ρ = makespan and similar P, d values. Once the execution is terminated, performance information of the new submission is stored.

Table 2.4 Monitoring commands. Complete metrics list (*) of perf: instructions, cycles, L1-dcache-loads, L1-dcache-load-misses, L1-dcache-stores, L1-dcache-store-misses, L1-icache-loads, L1-icache-load-misses, LLC-loads, LLC-load-misses, LLC-stores, LLC-store-misses, cache-misses and cache-references.

Tool	Command	Metrics
time	time -p	Real time, user time, system time
	/usr/bin/time -v -o	Maximum RSS, page faults
perf	perf stat -B -e metrics list(*) -o	List of 14 metrics: Instructions, cycles. 1^{st} and last level cache: loads, load misses, stores, store misses...
sar	sar -b	Av. & S.D. : Read and write requests issued per second to disk
vmstat	vmstat	Av. & S.D. : swapped in [MB/s], swapped out [MB/s], context switches [MB/s]
pidstat	pidstat -urd -p pid -h	Av., S.D. & med.: User CPU, system CPU %RSS, maximum RSS IO read [kB/s], IO write [kB/s]

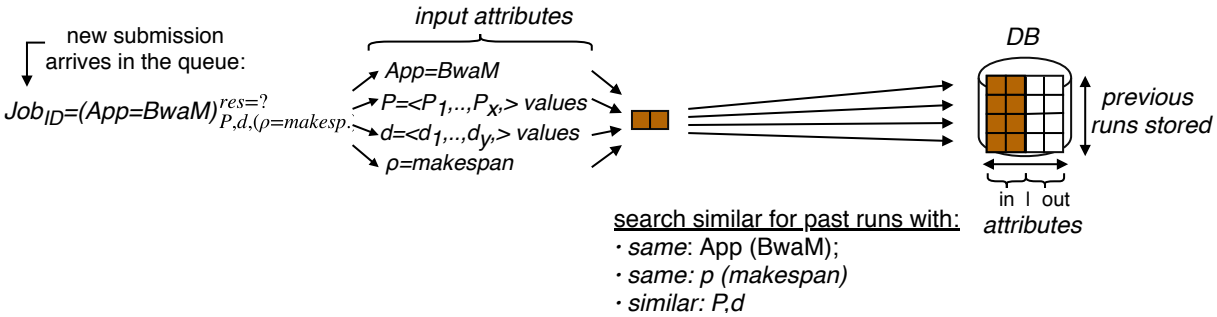


Figure 2.4 Searching for similar past executions in the database.

Some of the input and output attributes of each new record stored in the database are listed below:

- *Input Attributes*

  Application: name and version.

  Parameters: values of P1, P2, P3, P4, P5...

  Data: organism, file size, data type, format, number and average length of sequences.

  Resources: node architecture, memory, and PUs.

  Others: unique ID of the submitted task, output file format.

- *Output Attributes*

  User-performance: makespan, cost.

  Resources (memory): maximum RSS, Average RSS, Page faults.

  Resources (CPU): user time, system time, number of instructions, average Cycles Per Instruction.

  Others: size of the output file.

To determine the resources the new submission requires based on similar but not identical

executions carried out in the past, we considered including a predictor within the HBRM. Prediction however, is only feasible in those cases where data is inferential. In the following section we conduct inferential analysis to prove whether predictions can be applied to estimate the performance of future runs based on past ones.

2.2 Multivariate Regression Prediction Model

The goal of the Multivariate Regression Predictor is to estimate the performance of new tasks (application, parameters, data) based on similar executions stored in the database (same application, similar parameters and data characteristics).

Prediction models can only be applied in those scenarios where the data is inferential. Therefore, before attempting to develop a prediction model, we ran inferential analyses on a set of sample performance data.

2.2.1 Inference analysis

Even when repeating the same exact job execution with identical parameters, data characteristics, and resources, performance data such as makespan or cost may vary. Prediction is only viable if performance data is inferential.

Before building the prediction model, we generated multiple makespan data samples, and run a series of analyses to determine whether data is inferential or not. Since the same conclusions were extracted from all the samples, and the same explanation applies for all of them, only one sample case is shown. Sample data was generated by executing 12 identical instances of Hisat application, with identical parameter values, data characteristics and resources. The sample values obtained are represented within squares in Figure 2.5, ranging between 2673 and 2782, and averaging 2713.

The first analysis step consisted in determining whether makespan data is parametric. That is, whether it follows a normal distribution. To that end, the Shapiro-Wilk Normality Test was applied. A p-value of 0.037, smaller than 0.05, was obtained, indicated that data is indeed Normally Distributed. Once proven so, the confidence interval was calculated with the One-Sample T-Test. For the given sample, lower and upper limits equaling 2691 and 2734 seconds were obtained, respectively. The width of the confidence interval (43 seconds), represents a tiny percentage (1.65%) of the sample mean, thus proving that makespan data is inferential, and the prediction model viable.

2.2.2 Heteroscedasticity and Multicollinearity

Once proven the data is inferential, we reviewed two characteristics that are relevant to consider before attempting to develop a regression model:

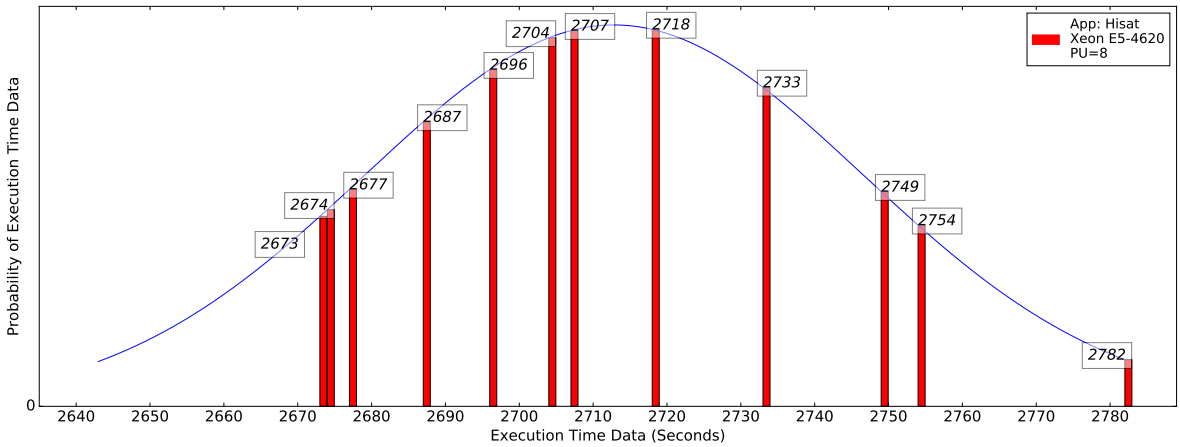


Figure 2.5 Makespans values (within rectangles and in seconds) of the sample chosen for the inferential analysis, and resulting Probability Density Function.

- Heteroscedasticity: in general, data is heteroscedastic if its standard deviations are unequal across a range of values. In regression analysis, heteroscedasticity is dealt with in the context of residuals of the error term. Data is heteroscedastic if the residuals change unevenly over the range of measured values. This is problematic because regression assumes that the residuals of a population have a somehow constant variance with respect to the regression line. A common explanation of heteroscedasticity is that the error variance changes proportionally with a factor, which may be a variable in the model. Heteroscedasticity can be detected with residual plots. Heteroscedastic data usually has a cone-like shape, as residual variability either decreases or increases with the dependent variable.

- Multicollinearity: it occurs when two or more predictor variables are highly correlated. That is, one can be used to predict the other. Multicollinearity generates redundant information, and wastes degrees of freedom.

2.2.3 Stepwise Variable Selection

The most common methods to determine the variables to be added in the model are: Forward Selection, Backward Elimination, and Stepwise Selection, which is also known as Forward Selection with replacement.

The forward methods start with a model with no predictor variables at all. With the t-test, they calculate the significance of potential candidate variables. The variable with the lowest p-value is added at each step, as long as it is lesser than the significance level α previously set by the user.

Backward elimination starts the model with all the variables included, and at each step removes the one with the highest p-value, as long as it is greater than the α level.

Every time a variable is added or deleted, the significance of the rest of the variables of the

model is altered. The problem with Forward Selection is that when a new variable is added, the significance of previously-added variables (which has been altered) is not checked. As a result, the final model may contain regressors of little relevance, which are merely redundant.

The Stepwise method overcomes this problem by re-computing the significance of previously-added predictors each time a new predictor is added. The Stepwise method, which yields similar results to Backward Elimination, has been chosen to develop the predictor.

When building the prediction equations, little prediction error is obviously sought. However, obtaining lesser prediction error when increasing the number of variables of a predictor doesn't always imply the prediction is better. Sometimes this phenomenon is due to the over-fitting of the model, which is the undesired modeling of the noise of data samples. Although little prediction error may be obtained on a sample, chances are the behavior of the sample doesn't represent that of the overall population of real observations. To prevent over-fitting, and evaluate the quality of the prediction model, the Adjusted R^2 coefficient has been calculated, as well as the relative prediction error.

Another metrics to determine the relationships between input and output variables is the Pearson Correlation Coefficient. However, this coefficient only calculates the amount of linear correlation between variables. Spearman's correlation gets around this problem by ranking output variables. That is, sorting them from lowest to greatest values, or the other way around. The problem with Spearman's is that only deals with monotonic functions, that is functions that either solely increase (monotonically increasing function) or solely decrease (monotonically decreasing function), with doesn't necessarily adjust to our data.

2.2.4 Prediction scenario and examples

At the very beginning, when the HBRM is implemented in a cluster, it starts with a limited amount of information in the database, such as that obtained with the characterization experiments. The HBRM monitors the performance of each user-submitted run, and stores the real performance information in the database. With this approach, the HBRM ensures a continuous and automated obtaining of knowledge, from which more and more accurate predictions will be generated.

We consider a scenario where users submit their workflow applications to clusters usually formed by numerous multi-core multi-socket nodes. They choose the application that will conduct the analysis, the values of the parameters, the datasets, and the performance criteria (assumed to be makespan). However, it is really difficult for users to determine the resources that the application needs. Not only the resource usage of applications can vary substantially upon the parameters and data combinations, but there is also a great amount of possible combinations to chose from. As depicted in Figure 2.6, users are unaware of the resources their applications need, and submit the workflow tasks to the cluster queue as follows: $\text{Job}_{\text{ID}}=\text{App}_{P,d,\rho}^{res=?}$. The cluster queue is then filled with tasks whose resource requirements are unknown, and the HBRM must determine, given also the resource availability, which resources must allocate to each task.

The resource availability is an important trait to be accounted for when allocating resources. The load of clusters varies over time, and consequently the availability of resources varies too. Both in heavily-loaded and barely-loaded scenarios, the HBRM will attempt to maximize the users' performance criteria, but will have to do so in a different manner. While in barely-loaded scenarios resources can generously allocated to a single task to maximize performance, in heavily-loaded ones where resources, a more-objective approach must be taken, since many tasks request resources which in turn may be scarce.

In order for the HBRM to dispose of enough information to allocate resources regardless of cluster availability, based on the performance information in the database, and the cluster status, the predictor generates for each job, (Job_{ID}=$App^{res=?}_{P,d,\rho}$) multiple performance predictions with different resources ($Pred_{ID}$=$App^{res=1}_{P,d,\rho}$,..,$Pred_{ID}$=$App^{res=x}_{P,d,\rho}$.), as depicted in Figure 2.6.

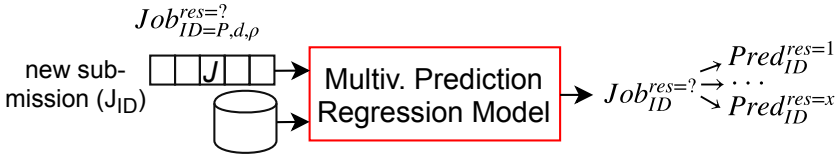


Figure 2.6 For each new submission in the queue, the Multivariate Prediction Regression Model of the HBRM generates multiple predictions with different combinations of resources.

Based on execution models and makespan predictions, the monetary cost of the execution can be easily derived. Due to that, this work focuses on the makespan as the performance criteria to minimize (ρ=makespan). The monetary cost of executing the submitted workflows was calculated by assigning fees to both the CPU model of the node and the peak memory consumed, by the hour of usage. Fares were estimated following the scale of Amazon's public EC2 instance pricing as of January 2018.

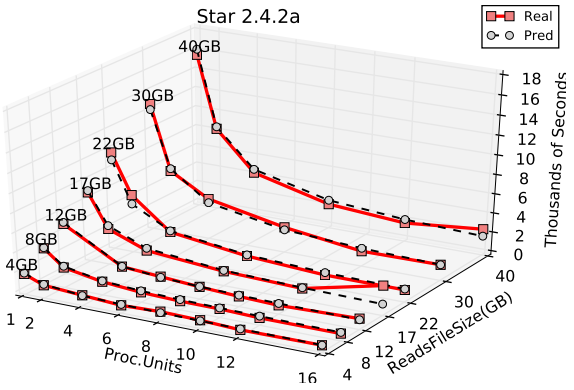
In Figure 2.7 we provide graphical representation of real makespan results alongside predicted results obtained with the Multivariate Regression Predictor of the HBRM, for two different cases of analysis.

Figure 2.7, (a), shows the real makespans (in red color) and predicted makespans (in black color) of Star application. The makespans in black color have been predicted in function of both the PUs and the reads file size. For this case, we obtained a mean relative error of 8.5%, and an Adjusted R^2 of 0.92.

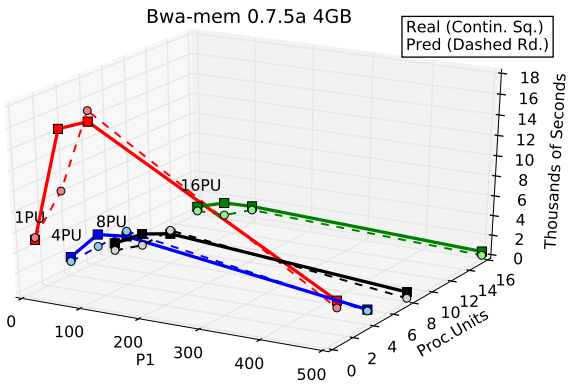
Figure 2.7, (b), shows the real makespan (in continuous lines) and the predicted makespans (in dashed lines) of Bwa-mem application. The makespans in dashed lines have been predicted in function of both the Parameter 1 (seed length) and the PUs. For this case, we obtained a mean relative error of 12%, and an Adjusted R^2 of 0.9.

As it can be observed from both graphical representations, and the mean relative error and R^2, the predicted makespan results prediction results remain close to the real ones.

For Star's case, the prediction Equation 2.2 was generated, where: α=3.6*10^{-7}, β=1.03*10^{-8}, and γ=219.4. For Bwa-mem's case, the prediction Equation 2.3 was generated, where: α=-0.88,



(a) Real and predicted makespans of Star, in function of the amount of PUs and the size of the reads file.



(b) Real and predicted makespans of Bwa-mem, in function of the P1 (Parameter 1, seed length), and the amount of PUs.

Figure 2.7 Real and predicted makespans of Star and Bwa-mem applications, each in function of two predictor variables.

β=0.016, $\gamma = -3.3 * 10^{-5}$, and δ=8.6.

$$PredTime = \alpha * (Size/PUs) + \beta * Size + \gamma \tag{2.2}$$

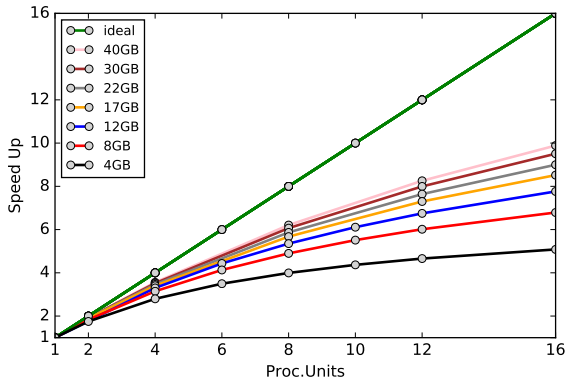
$$PredTime = e^{\alpha * ln(PUs) + \beta * P_1 + \gamma * P_1^2 + \delta} \tag{2.3}$$

In shared environments, multiple tasks may simultaneously attempt to use the same resources. Some of them are allocated resources, as others wait for them to be released. Cluster efficiency has to be taken into account to prevent, among other things, over-allocation of resources, which would cause them go wasted as other jobs wait. Furthermore, chances are jobs aren't able to reserve all the desired resources, such as the whole node. On top of that, jobs may not even get to run with as many PUs as the scalability threshold marks, $numPUs_{MaxSpeed}$, as they would in exclusive mode. Instead, they may have to be executed with less PU, i.e. $numPUs_{LowSpeed}$, yielding longer makespans (makespan penalty).

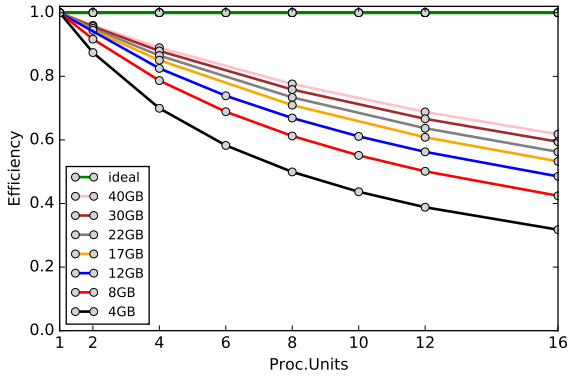
To properly deal with the trade-off between: (i) having to wait for resources such as PUs to be freed to run by maximizing as much as possible applications' performance, and (ii) running as soon as possible (without having to wait) using the resources currently available, the HBRM calculates both the speed up and efficiency predictions. The speed up and the efficiency predictions can be obtained from the makespan predictions generated with multiple resources, such as a wide range of PUs in each node.

To illustrate that, we show in Figure 2.8(a), and Figure 2.8(b), (respectively) the predicted speed ups and the predicted efficiencies of Star application that the predictor has obtained from the makespan predictions of Figure 2.7(a).

The scalability threshold of Star with the chosen parameter values and resources, ranges between 12 PUs and 16 PUs depending on the file of the dataset. To simplify, we could state that



(a) Star's predicted speed ups.



(b) Star's predicted efficiencies.

Figure 2.8 Predicted speed ups and efficiencies for Star application.

for Star (with the given parameter values and 40GB), the $numPUs_{MaxSpeed}=16$.

Also, in Figure 2.8(b), we can see that efficiency ranges between 0.3 and 0.6, averaging 0.5.

Nonetheless, by taking a closer look to the predicted speedups of Figure 2.7 (a), one may notice that if instead of allocating to Star $numPUs_{MaxSpeed}=16$, we allocated $numPUs_{LowSpeed}=12$, little makespan penalty (time difference between using 12 or 16 PUs), would be inflicted. Additionally, by allocating 12 PUs instead of 16, the efficiency is increased.

Another important fact, is that by considering the speed ups obtained from different-resource predictions, the portion of node resources saved, such as PUs, can be used by other applications. Thus, by allocating Star 12 PUs instead of 16 PUs, 4 PUs for the given could be released and given to other applications in the queue, which could be executed at once, drastically reducing the queue waiting times and increasing resource utilization.

This finding is of paramount importance for improving the overall performance of clusters, especially of those mainly hosting bioinformatics workflow applications.

Most bioinformatics applications, as reviewed in Chapter 1, due to their shared-memory paradigm, and a poor memory management (addressed in Section 2.3), show a low scalability. By considering this trait, and with scalability predictions, the significant amounts of PUs and other node resources such as memory, can be saved with little makespan penalties. This opens the door for developing a resource sharing approach, as these node resources that are saved can in turn be harnessed by other applications that would otherwise remain in the queue.

In the next section, we review the resource of bioinformatics applications in order to determine, based on their characteristics and predictions, how to develop an approach to share node resources.

2.3 Multicriteria Scheduler

The third block of the HBRM is the Multicriteria Scheduler, which receives the multiple performance predictions generated by the predictor for each workflow tasks in the queue. While the main concern of the predictor was to adjust resource allocation to the actual needs of applications, and by doing so leaving more node resources free (such as PUs) for other tasks, in this section we focus on how to properly allocate these recently-freed PUs. In other words, we at finding a proper resource sharing strategy.

Most clusters are not exclusively used but shared among many tasks simultaneously running in the same nodes, which are then said to be multiprogrammed (DP>1). Increasing the DP of the nodes is usually beneficial for overall performance. For instance, node multiprogramming allows for multiple tasks in the cluster queue to be simultaneously executed, shortening waiting times and turnarounds. Additionally, in cases where applications show poor scalability, by adjusting resource allocation to applications needs (such as suggested by the predictor of the HBRM), PUs can be freed. In other approaches that don't consider applications' resource needs, such as the ones followed by current RMS on clusters, these PUs would be allocated anyways, despite of being barely used by the poorly scalable application. With the proposed approach, these PUs otherwise wasted can be used by other applications.

For instance, given a cluster-submitted queue including Blast and Phyml application tasks of Table 2.1, and a series of nodes with 64 PUs of PUs, by being aware in advance of the poor scalability of the given Blast application version (2.2.26), which hardly exceeds 12 or 16 PUs, would carry significant benefits on overall makespan and resource utilization. Thus, instead of allocating the whole set of node PUs, or even a large amount of node PUs, to Blast, we could adjust the amount of PUs to the real needs of Blast (around 16 PUs), and free the rest of PUs (around to 48), to allocate them other queued tasks such as Phyml.

Node sharing has doubtless benefits for the system's performance. However, it usually comes at the cost of slowing down the makespans of applications sharing the node, compared with their respective exclusive-execution cases. Thus, the trade-off between increasing the DP of the nodes and the resulting makespan slowdown must be accounted for. The slowdown depends not only on the node specifications but also on applications' characteristics, and some combinations of applications yield significant greater slowdowns than others. Resources must be properly shared in order to make the most of them, and prevent slowdowns from soaring up to a point where computer power is squandered and little or no benefits are seen from node sharing. Picking up the idea of the paragraph above, not only we can release the 48 PUs to give it to another application, but we can also establish *which* application makes the best candidate for occupying those 48 PUs alongside Blast to minimize resource competition and thus slowdown.

In this section we address the problem of scheduling a series of workflow tasks submitted to a cluster queue by using the performance predictions that allow us to adjust the amount of PUs to the jobs' needs. We determine a proper approach to share node resources that allowx us to

tell which combinations of applications make good candidates for same-node execution so that potential performance issues stemming from resource competition are minimized. To figure out a proper scheduling approach that considers node multiprogramming, we analyzed applications' individual performance and identified their resource usage patterns. Based on this information, we designed a series of experiments to observe the slowdown yielded by shared-node execution of different combinations of applications.

Next, we developed three scheduling algorithms for shared clusters hosting bioinformatics workflow applications. Given scenarios involving a series of ready workflow tasks in a submission queue, and partially-loaded node resources such as PUs, based on performance prediction performance in exclusive mode and slowdown information, the algorithms determine how to schedule applications.

The first algorithm developed, the Slowdown-Aware (SA) scheduling algorithm, focuses its operation mode on two major lines. On one hand, how to adjust the amount of PUs to applications needs to leave PUs free for other applications causing little makespan penalty. On the other, by determining which combinations of tasks are to run in the same node to minimize slowdown and improve the overall performance.

The second algorithm, the Biobackfill Scheduling algorithm, is a variation of the SA algorithm that backfills applications in scheduling gaps.

The third algorithm, Slowdown-Aware File-Placement (SA) Scheduling Algorithm, is an extension of the first algorithm that considers the location of the workflows files in the different hierarchy levels of the cluster to minimize latencies stemming from file seek and transfer times. The SAFP algorithm belongs to a new research area that we have been lately working on in cooperation with fellow researchers' past work.

2.3.1 Resource usage model of bioinformatics applications

The resource usage of bioinformatics applications must be understood in order to address the problem of determining how to distribute a series of applications among shared resources. For the given case, and as customary in the bioinformatics area, we considered applications following a shared-memory paradigm. That is, executed in a single node at a time by exploiting thread-level parallelism. Therefore, we cast the focus on show resources of a node (mainly PUs) can be properly shared among multiple applications.

The bulk of applications included in Table 2.1 is formed by read mappers, which conduct one of the most time-consuming stages within the genomic analysis pipelines. Read mappers manage huge amounts of biological file sizes. In concordance with the rest of the document, we focus on genome files, although the same reasoning applies for aminoacids. File sizes have a great impact on bioinformatics applications' performance, and must be considered. Mappers mainly have two kinds of input files: the indexed genome reference files, usually of a size that fits into memory, and the genomic sequence reads files, generally of a much greater size.

Mappers find the exact position in the genome of the sequences within the reads file. One of the most popular read mapping applications is Bwa, and most mappers follow its strategy to transfer the reads files from memory to CPU. The reads file is not analyzed in one go, but divided into parts called chunks and distributed in a round robin fashion to threads for parallel analysis. A double buffer scheme is implemented to do so, minimizing transfer latencies by simultaneously analyzing some chunks as others are brought from memory.

The ratio of memory operations versus other operations is high in some mappers [24], as large volumes of data are continuously transferred from memory. Furthermore, the memory access pattern of these applications is highly random. This poses serious issues, since adds uncertainty to the process of deciding which data block is going to be fetched next. To acquire requested data, the hardware prefetcher usually exploits data locality, principle based on the assumption that once a data block is fetched, there is a high probability the next block will be requested. However, to implement this mechanism, sequential data access patterns, as opposed to random patterns, must be detected.

Fetching the reference sequence and mapping reads involves a high degree of randomness, as the mapped positions of a given read seed cannot be statically determined in advance [55]. Sequential prefetching can't be applied, and the cache can't be kept pre-loaded with presumably upcoming data blocks. As a result, the amount of a significant amount of last-level cache misses arise.

The amount of memory to be transferred and often-failing cache speculation tend to saturate the memory bandwidth and increase the latency. This phenomenon becomes more notorious as the number of threads is augmented [23]. While waiting for data, the CPU is low-utilized or stalled, and cycles are wasted. Read mapping applications are memory-bound, and performance limitations shown when increasing the number of concurrent execution threads cause them to have a low scalability [40, 41, 62]. As a result of the memory bandwidth saturation and latency, CPUs are barely used, waiting for memory requests to be solved, as other jobs waiting in queue can't access them.

2.3.2 Resource-sharing approach

By analyzing the resource usage of read mappers, we have determined the main reasons behind their low scalability, as well as their performance boundedness.

The poor memory management of read mappers prompts the execution threads to generate a considerable amount of memory requests, carrying out large volumes of data that end up saturating the bandwidth even with a relatively low number of PUs, which are stalled waiting for data to arrive.

The resource usage of mappers, generally labeled as memory-bound, contrasts with that of other bioinformatics applications included in Table 2.1, the phylogeny tools. These applications compute trees commonly following Maximum Likelihood methods that are heavy on CPU [58].

Due to that, phylogeny tools are usually branded as CPU-bound applications.

Mappers' low scalability raises the challenge of how to use the remaining PUs of the node, or how to properly increase the DP of the nodes, so that slowdown is reduced to minimum.

In this work, slowdown is quantified as the average percentage of makespan increase inflicted to all applications sharing a node, compared with their respective exclusive-execution cases. As stated before, the slowdown depends not only on the specifications of the multiprogrammed node, but also on the characteristics and resource usage carried out by each application running on it.

Thus, identically-bound applications will be much more likely to increase resource competition and interfere on each other's executions than differently-bound applications. For instance, multiple memory-bound applications will likely attempt to recurrently access the memory at the same time. One application may be granted access while the other may be prompted to wait for the first to release it, as the CPUs remain stalled. Conversely, applications frequently using different resources, such as memory-bound mappers and CPU-bound phylogeny tools, could potentially make good candidates for same-node execution, as they are likely to mainly request different resources.

Combinations of applications with different resource usages yield significantly lesser slowdown than combinations of similar applications, and are suitable to share nodes. Hence, we propose scheduling combinations of applications onto multiprogrammed nodes based on their resource usage.

2.3.3 Slowdown Experiments

To determine the amount of slowdown yielded when multiple tasks share the same nodes, we conducted a series of experiments in which characterized applications from Table 2.1 where simultaneously executed in different nodes of the cluster of Table 2.3.

Aside from applications, the slowdown depends on multiple factors, such as the parameter values, input data characteristics, node characteristics ... To simplify the experiments and focus on the benefits of the approach, in all slowdown experiments, each application has been executed with the same dataset, default parameter values, node (AMD), and an amount of PUs matching their maximum speed: $\text{numPUs}_{\text{MaxSpeed}}$.

The slowdown experiments are exemplified with DP=2. Thus, we generated all possible combinations of two applications, and executed them all on a cluster node with $\text{numPUs}_{\text{MaxSpeed}}$. The makespans of each application were monitored, and compared with the respective makespans (obtained in the Application characterization experiments) with the same applications, parameters, data, and resources. The resulting slowdowns of all the combinations of applications are shown in Table 2.5. The slowdown depends not only on applications but also on their parameters and data. However, to keep the focus on the addressed point, which is to show the benefits of multiprogramming by accounting for the resource usage of the applications, we simplify the

slowdown information of Table 2.5, and refer only to the applications.

Table 2.5 Slowdowns obtained when running top-row alongside left-column applications simultaneously on the same cluster node.

	Blast	**BwaM**	**Bowtie**	**BwaA**	**Hisat**	**Star**	**Soap**	**Phyml**	**Mrbayes**	**Fasttree**	**Raxml**
Blast	8%	%	%	%	%	%	%	%	%	%	%
BwaM	18%	2%	%	%	%	%	%	%	%	%	%
Bowtie	6%	12%	10%	%	%	%	%	%	%	%	%
BwaA	16%	23%	21%	3%	%	%	%	%	%	%	%
Hisat	18%	2%	16%	18%	8%	%	%	%	%	%	%
Star	21%	19%	30%	6%	18%	31%	%	%	%	%	%
Soap	6%	3%	15%	2%	6%	13%	10%	%	%	%	%
Phyml	2%	6%	7%	11%	3%	1%	0.2%	12%	%	%	%
Mrbayes	3%	6%	2%	3%	3%	5%	4%	27%	6%	%	%
Fasttree	5%	1%	2%	0.5%	7%	0%	0.5%	21%	4%	6%	%
Raxml	3%	3%	4%	2%	4%	7%	1%	16%	4%	6%	8%

Consulting the slowdown table we can assess, given a queue with multiple applications to be combined for same-node execution, which combinations of applications make best candidates to share nodes. Applications generally requesting similar resources, such as BwaM+BwaA will usually yield a high slowdown (23%), and should be avoided by the scheduling algorithm. Conversely, combinations of applications generally requesting different resources, such as BwaM+Fasttree should be sought by the scheduling algorithm, as they minimize slowdown (1%). This principle is graphically shown in Figure 2.9. Given a cluster node hosting BwaM (Bwa-mem) application, which has been allocated an amount of PUs adjusted to its needs, and a submission queue containing all applications in the set, opting for fasttree to be executed alongside BwaM would entail a 22% slowdown reduction compared to opting for BwaA (Bwa-aling).

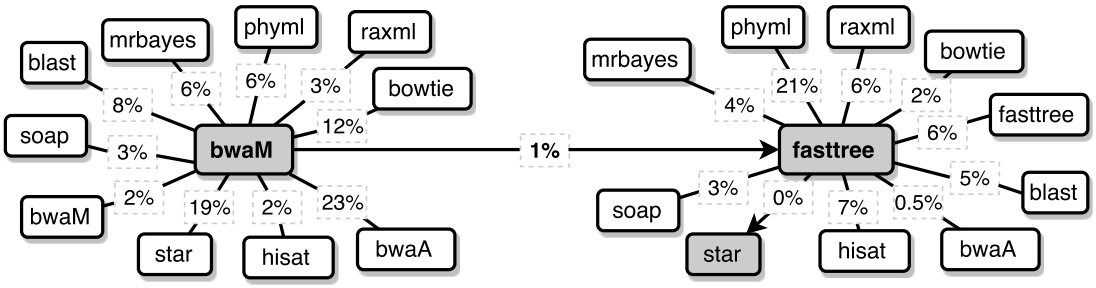


Figure 2.9 Up to 22% slowdown reduction can be obtained by scheduling (BwaM+Fasttree=1%) for same node-execution, compared with (BwaM+BwaA=23%).

Augmenting the DP is beneficial as it improves overall performance, increasing resource utilization and reducing waiting times. However, nodes have a maximum DP, a point from which augmenting the number of applications is no longer advantageous, bringing about an abrupt slowdown increase.

The maximum DP of a node depends on the node hardware characteristics, the applications, parameters, data and PUs selected. Optimizing the DP of the nodes for each execution case is a complex issue beyond the scope of this work. Instead, a DP=2 has been chosen since it allows the improvements and mechanism of the proposal to be exposed. In order to reflect the complexity of calculating the maximum DP, a summarized example, which can be extrapolated to any other case, is provided below. The example has been conducted by selecting 8 applications of Table 2.1, a specific number of: PUs for each application ($\text{numPUs}_{\text{MaxSpeed}}$), parameters, data, and the AMD node of Table 2.3. To determine the maximum DP, all combinations of different amounts of applications were executed on the cluster, and the average slowdowns calculated each time. First, all combinations of 2 applications (DP=2) were executed. Next, all combinations of 3 elements (DP=3), 4 elements (DP=4), and 8 elements (DP=8), were respectively executed. The average slowdown evolution with combinations of different amounts of applications (DP=2, DP=3, DP=4 and DP=8) is depicted in Figure 2.10. Results of this given example indicate that little slowdown increase is spawned when increasing the DP from 2 to 3 applications (1.4%), and from 3 to 4 applications (4%), respectively. Thus, the four applications with the given conditions of parameters, data and PUs can be executed in the given node with little consequences on performance. Nonetheless, when increasing the DP from 4 to 8, slowdown soars up to 65%.

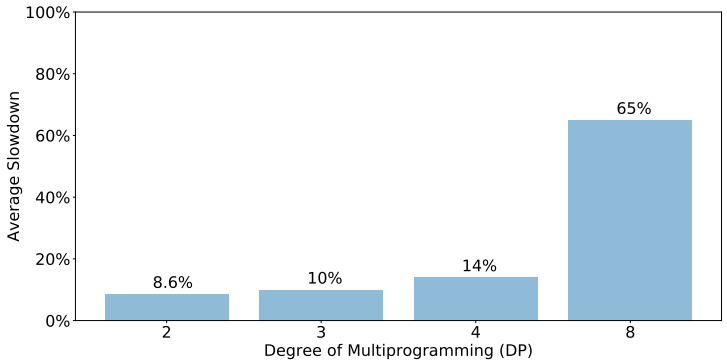


Figure 2.10 Average slowdowns yielded when increasing the DP, for a given set of applications and resources.

2.3.4 Scheduling Algorithms developed

In this work, we have developed three different scheduling algorithms for bioinformatics workflow applications running on shared clusters. The three algorithms consider the performance predictions to allocate PUs, and the slowdown information to multiprogram the nodes. All three algorithms consider for allocation a broken-down list of workflow tasks in ready state.

2.3.4.1 Slowdown-Aware (SA) algorithm

The first scheduling algorithm included in the HBRM is the Slowdown-Aware (SA) scheduling algorithm, shown in Algorithm 1. The algorithm has been developed to schedule a queue of

bioinformatics workflow tasks submitted to shared clusters.

The scheduling algorithm receives three inputs:

- LApps: List of Applications to Schedule
- LPred: List of Predictive Times of Applications in Exclusive Mode, for PUs in node, for app in LApps.
- LCompat: List of Compatibility Slowdowns of Applications, for app in LApps.

The algorithm outputs LPrio, a list of tasks sorted by priority and with resources allocated. Within LPrio, combinations tasks are allocated for same node execution so that slowdown is minimized. Also, tasks are allocated an amount of PUs adjusting to their needs (upon application, parameters, and data).

The pseudo code included in this document corresponds to the one included in our first publication (PBio, May 2017). The code has been modified ever since.

Algorithm 1: Slowdown-Aware (SA) scheduling algorithm

```
Inputs : LApps: List of Applications to Schedule;
         LPred: List of Predictive Times of Applications in Exclusive Mode, for PUs in node, for app in LApps;
         LCompat: List of Compatibility Slowdowns of Applications, for app in LApps;
Output : LPrio: List of Applications Sorted by Priority;
1  Read LApps;          // Read predictions of apps in LApps with different PUs, nodes, from LPred
2  while apps in LApps do
3      Sort List of Applications to Schedule (LApps) by Time Descending;                  // Longest to Shortest
4      ApLong = First in LApps;
5      Sort List of Compatibility Slowdowns of Applications (LCompat) for ApLong by Slowdown Descending; // Compat apps
         of each app
6      ApComp = First in LCompat for ApLong ;                                  // ApLong's most compat app
7      SelectedApps = ApLong & ApComp;
8      MaxRes = PUs for Max SpeedUp for app in SelectedApps ;                          // Obtained from LPred
9      Read Resource Status;
10     if IdleNodes in cluster then
11         if AvailRes > MaxRes then
12             SelectResources = MaxRes for SelectedApps;
13         end
14         else
15             SelectResources = Combination of PUs for SelectedApps minimizing sum of LPred times;
16         end
17         UpdateLists(SelectedApps);
18     end
19     if LoadedNodes in cluster then
20         SelectedApps = LoadsMostCompatApp (for app in LApps) ;      // Get app in LApps most Compat with
             node's Load
21         if AvailRes > MaxRes(SelectedApps) then
22             SelectResources = MaxRes for SelectedApps;
23         end
24         else
25             WaitMaxResRelease = (PredTimeApp – CurrentTimeApp) for App in Load;
26             Wait&RunMaxRes = WaitMaxResRelease + PredTime running with MaxRes;      // Wait MaxResRelease,
                 run with MaxRes
27             RunAvailRes = PredTime running with AvailRes;                  // Run at once with AvailRes
28             if RunAvailRes < Wait&RunMaxRes then
29                 SelectResources = AvailRes for SelectedApps;
30             end
31             else
32                 SelectResources = MaxRes for SelectedApps;
33             end
34         end
35         UpdateLists(SelectedApps);
36     end
37 end
38 return List of Applications sorted by Priority (LPrio);
39 Function UpdateLists (SelectedApps)
40     Remove SelectedApps from LApps;
41     Add SelectedApps to LPrio;
42     Go to Next App;
```

2.3.4.2 Biobackfill algorithm

The second scheduling algorithm included in the HBRM is the Biobackfill scheduling algorithm, shown in Algorithm 2. The algorithm has been developed to schedule a queue of bioinformatics workflow tasks submitted to shared clusters.

The scheduling algorithm receives four inputs:

- LJobs: List of queued Jobs ($\text{Job}_{ID=1,..,q}$=app,param,data)
- LPred: List of Perf.Preds. for PUs in node, for job in LJobs
- LDep: List of Dependency Status for job in LJobs

- LSlow: List of Slowdowns, for job in LJobs.

The algorithm outputs LPrio, a list of tasks sorted by priority and with resources allocated. Within LPrio, combinations tasks are allocated for same node execution so that slowdown is minimized. Also, tasks are allocated an amount of PUs adjusting to their needs (upon application, parameters, and data).

The pseudo code included in this document corresponds to the one included in our second publication (COMPLEXIS, March 2018). The code has been modified ever since.

Algorithm 2: Bio-Backfill Scheduling Algorithm

Inputs : LJobs: List of queued Jobs ($Job_{ID=1,...,q}$=app,param,data)
LPred: List of Perf.Preds. for PUs in node, for job in LJobs
LDep: List of Dependency Status for job in LJobs
LSlow: List of Slowdowns, for job in LJobs
Output : LPrio: Priority-Sorted List of Jobs

```
1  while jobs in LJobs do
       ;                                          // For each Job_ID, in different res.(node,PU)
2      Calculate Perf.Preds. (LPred);             // times,speedups,effi.
3      Calculate Slowdowns (LSlow);               // dif.combis of jobs
4      Calculate Job Dependencies (LDep);         // Pred.  WaitDep
5      MaxRes_ID = PUs for Max SpeedUp ;          // for Job_ID
6      LowRes_ID = Range PUs below MaxRes_ID ;    // t.penalties
7      Read Resource Status;                      // res avail, nodes' load
8      if IdleNodes in cluster then
9          Job_LRe = Longest Ready Job_ID
10         Slow_Re = min Slow (Job_OthersRe,Job_LRe)
11         Wait&Slow_NoRe=Slow(Job_OthersNoRe,Job_LRe)+WaitDep
12         SelectedJobs= Jobs with min (Slow_Re,WaitSlow_NoRe) if ResAvail(node) < MaxRes(SelectedJobs) then
13             SelecRes = PUs minimizing sum of LPred times
14         end
15         else
16             SelectRes = MaxRes for SelectedJobs
17         end
18     end
19     if LoadedNodes in cluster then
20         Slow_Re= MinSlow(Job_Load,Job_ID.Re)
21         Wait&Slow_NoRe=min (Slow(Job_Load,Job_ID.Re)+WaitDep)
22         SelectedJobs = Jobs with min(Slow_Re; Wait&Slow_NoRe)
23         if ResAvail(node) > MaxRes(SelectedJobs) then
24             SelecRes = MaxRes for SelectedJobs
25         end
26         else
27             Time_AvailRes = PredTime Job_ID,AvailRes
28             Wait&Time_MaxRes = Wait_MaxResFree + Time_MaxRes
29             SelectedRes=min(Time_AvailRes,Wait&Time_MaxRes)
30         end
31     end
32 end
33 return List of Jobs sorted by Priority (LPrio)
```

2.3.4.3 Slowdown-Aware File-Placement (SAFP) algorithm

The third scheduling algorithm included in the HBRM is the Slowdown-Aware File-Placement (SAFP) algorithm. The SAFP algorithm is the integration of the SA algorithm (Algorithm 1) plus the FP algorithm (Algorithm 3). However, throughout this work we refer to Algorithm 3 as the SAFP algorithm.

The SAFP algorithm includes the considerations of the SA algorithm (prediction, slow-down...), plus the consideration of the different memory hierarchy levels of the cluster to place workflows' files. The file-placement (FP) portion of the SAFP algorithm has been developed in collaboration with another research line of our research group. The objective was to merge both lines (SA and FP) in an integrated proposal (SAFP), and evaluate its performance.

Bioinformatics workflows are complex pipelines made of multiple tasks executed following a series of dependencies. Each workflow tasks may take as input one or more files of considerable size, which contain the biological data. To conduct the analyses due, tasks run their algorithms on input files by continually requesting data chunks, which may be placed in the different memory hierarchy levels featuring in the system, such as a shared NFS, nodes' local disks or nodes' RAM disks. In this context, a significant flow of data is sent from these levels to CPU to carry out analyses. Depending on the intrinsic applications' algorithms, such as those of read mappers, files and sequences within must not only be accessed at great speeds, but the speed soars as more execution threads are requested.

The portion of time some data-intensive bioinformatics applications destine to transfer data back and forth from its original storage placement to CPU may represent a significant percentage of the overall execution time. Furthermore, data-related processing operations may become one the performance-bounding factors of these applications.

When executed on computing nodes, tasks generate result files, the size of which may surpass that of the input files. If the result files are also the input files of at least another (next) workflow applications, they are referred to as intermediate files. The onset of the next tasks must be on hold until the intermediate files of the previous tasks are generated. Once so, the dependencies are solved and the next applications are ready to start execution.

The location where workflow-requested dataset files are placed plays a major role on the resulting task performance of those applications dealing with large data volumes. Other applications however, are not so dependent on file pre-processing considerations in order to attain good performance. Examples are those dealing with more modest dataset file sizes, as well as others whose main complexities challenge resources different from memory, such as CPU-bound phylogenies.

Naive placement of workflow files may incur in longer latencies, seek and transfer times by some tasks, lengthening their execution times, and postponing the onset of others. Instead, placing workflow files by considering the performance consequences may significantly contribute to shorten the amount of time some applications dedicate to fulfill file management and processing needs, as well as reducing the waiting times the next applications must wait in order for their dependencies to be solved.

In order to improve the overall makespans applications, we have developed a strategy that adequately distributes workflows' requested files in the different memory hierarchy levels by accounting files characteristics. With this approach, files are placed in those storage units yielding

best latencies, seek and transfer times, in such a way that overall workflow makespan is further improved, and memory utilization is maximized. To that end, we extended the SA algorithm, shown in Algorithm 1, and developed the Slowdown-Aware File-Placement (SAFP) algorithm, shown in Algorithm 3.

The SAFP algorithm assumes that at the beginning of the execution, all files required by the workflow tasks that will be executed are placed in a single NFS of theoretically unlimited size, which is shared among all nodes on the cluster. Also, it is assumed that there are two other storage units on the cluster: each node has its own local disk, as well as and local RAM disk.

The SAFP algorithm places workflows files in the different memory hierarchy levels upon two considerations. The first one is the file size. Three file sizes are considered in Algorithm 3: small (fileSize $<$ 1MB), medium (1MB $<$ fileSize $<$ 1GB), and big (fileSize $>$ 1GB). The second consideration is whether files are shared (used by more than one task in the workflow), or non-shared (used by a single workflow task).

Usually, faster memory levels are smaller and more expensive. One of the ideas behind the algorithm is to place as many files as possible in the fastest memory levels, so that the performance of as many tasks as possible can benefit from it. Bigger files occupying more significant proportions of the overall memory capacity are sent to slower-though-greater memory levels. The other idea is that exceptions should be made for files to be used more often than others, i.e. shared files. Thus, not-so-small shared files may be allowed in faster storage levels.

As can be seen in Algorithm 3, small and shared medium files are sent to the local RAM disk of the node where the execution takes place. Non-shared medium files and shared big files are sent to the local disk of the node hosting the execution. Finally, non-shared big files are kept in the NFS storage system.

Algorithm 3: File-Placement (FP) algorithm portion, attached to the SA algorithm to form the Slowdown-Aware File-Placement (SAFP) Scheduling Algorithm

```
Inputs : requiredFiles: List of Required files of each task, and size of each file;
         Memory Hierarchy Levels and specifications: latencies, transfer times, seek times, sizes;
         Task to be executed and execution Node;
Output : aggregatedFileTime ;                                  // time to manage all files a task requires
1  boolean fileIsShared ;                                  // true if a file is used by > 1 task of the workflow
2  fileSize=size of the file in Bytes;
3  aggregatedFileTime=0 ;                                                  // agregated fileTime of a task
4  for file in requiredFiles do
5      if (file is inputFile) then
6          if ((fileSize < 1MB) or (1MB<fileSize<1GB and fileIsShared==true));   // file belongs in local Ramdisk
7           then
8              if (file is already in Ramdisk) then
9                  fileTime=ramSeek+ramLatency;
10             end
11             if (file is not yet in Ramdisk) then
12                 fileTime=NFStransfer+NFSseek+NFSlatency+ramSeek+ramLatency;
13             end
14         end
15         if (((1MB<fileSize<1GB) and (fileIsShared==false)) or (fileSize>1GB and fileIsShared==true));  // file belongs
             in local SSD disk
16          then
17             if file is already in local SSD then
18                 fileTime=SSDSeek+SSDLatency;
19             end
20             if (file is not yet in local SDD) then
21                 fileTime=NFStransfer+NFSseek+NFSlatency+SSDSeek+SSDLatency;
22             end
23         end
24         if (fileSize>1GB and fileIsShared==false) ;                                  // file belongs in NFS
25          then
26             fileTime=NFStransfer+NFSseek+NFSlatency;
27         end
28         aggregatedFileTime=fileTime +aggregatedFileTime;
29     end
30 end
31 return aggregatedFileTime;
```

.

Chapter 3

Adapting Workflowsim to implement the scheduling algorithms of the HBRM

In this section we describe the modifications done on Workflowsim simulator to implement the HBRM on large clusters generated with Workflowsim. In Section 3.2, we explain the how the Workflowsim's node resources are configured to recreate a cluster, and how we multiprogram these nodes. In Section 3.3, we list and describe the modifications necessary on Workflowsim in order to be able to implement the three scheduling algorithms of the HBRM: SA, Biobackfill, and SAFP.

The HBRM has been built to be implemented on large clusters hosting bioinformatics workflow applications. Before doing so, the HBRM must be validated. The validation of the HBRM will be conducted in Chapter 4 in different clusters, where the performance HBRM and its three scheduling policies (SA, Biobackfill, and SAFP) will be compared with other state of the art scheduling policies. Part of the validation experiments will be conducted in physical clusters such as the one available for this work, described in Table 2.3.

However, validation in physical clusters comes with numerous hurdles, such as the economic cost of having to purchase and host the computing platform, as well as having to supply it with power over long periods of time such as months or years. Those expenses increase for each node added in the platform, conditioning its size and the environment with which the validation is done. Furthermore, physical clusters already come with default RMS which include sets of scheduling policies that are selected by the platform administrator.

To overcome these limitations, in Chapter 4, we will also conduct the validation of the HBRM in simulated clusters, using Workflowsim. Among the advantages of using simulation to validate the HBRM, we can find the possibility to validate the HBRM on large clusters with a wide range of nodes, and extend the comparison of the HBRM to other scheduling algorithms which may not be included in the default RMS of the physical cluster, handled by the platform administrator.

Workflowsim can host different computing platforms, such as clouds and clusters. The HBRM will be validated on large clusters with. To recreate large clusters with the simulator,

their resource specifications must be configured accordingly. In Section 3.2, we explain how we configured the resources of Workflowsim to recreate large clusters.

Workflowsim is a widely-used simulator for executing workloads on large clusters which includes a series of state of the art scheduling policies, such as Min-Min, Max-Min, HEFT or Round Robin. Also, it also offers users the possibility to implement their own scheduling policies. We took advantage of this feature to implement in it the algorithms of the HBRM: SA, Biobackfill, and SAFP. However, we found out that some essential considerations needed to implement the HBRM algorithms were missing in Workflowsim. In order to to proceed with the implementation, some of the main blocks of Workflowsim had to be modified. The modifications conducted in Workflowsim for the HBRM algorithms are explained in Section 3.3.

Once the Workflowsim blocks are adapted to the needs of the HBRM's SA, Biobackfill, and SAFP algorithms, it must be proven that no errors have been introduced in the simulation tool. The validation of the correctness of the modifications introduced in Workflowsim is done with a series of experiments that we will conduct in Section 4.3.

Before proceeding with the modifications conducted in the simulator, we review the main blocks of the architecture of workflowsim

3.1 Workflowsim architecture

In order to test the performance of the developed scheduling policy, we employed the Workflowsim simulator, an extension of Cloudsim that includes an extra layer for managing workloads with dependencies. Workflowsim is an open-source tool widely adopted by the scientific community, that manages DAG-modeled workflows in XML format. It offers a framework for modeling and simulating cloud and cluster environments: datacenters, virtual machines...

The architecture of Workflowsim is depicted in Figure 3.1, and consists of four main blocks, vertically disposed at the left-hand side Figure 3.1, and reviewed from top to bottom.

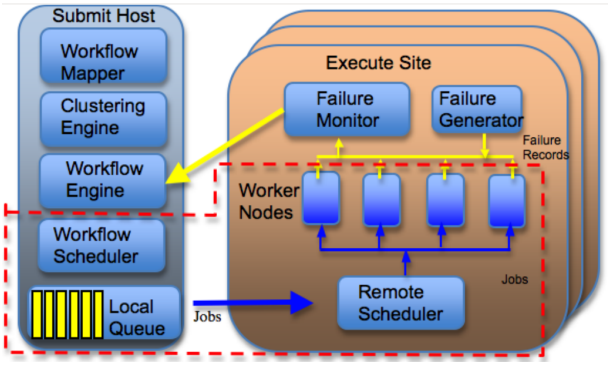


Figure 3.1 Architecture of Workflowsim simulator.

The Workflow Mapper includes a workflow planner that has a view of the whole workflow: tasks, dependencies.. and a workflow parser that reads the XML file, parses it, and creates a list

of tasks. The list of tasks is sent to the Clustering Engine, which merges them into jobs if the user specifies so: horizontally, vertically, randomly... From that moment on, tasks are referred to as jobs even if no merging takes place. The Workflow Engine ensures jobs are submitted to the Workflow Scheduler once they are ready, releasing free tasks in each scheduling iteration.

Based on the user-specified (or developed) scheduling algorithm, the Workflow Scheduler matches tasks to worker nodes. Unlike Cloudsim, Workflowsim includes support for dynamic workflow algorithms, aside from the static ones. Static or planning algorithms can bind any task to any resource, although the actual execution order will depend on the resource availability. Planning algorithms assign worker nodes at the Workflow Planner stage. Once the job reaches the remote scheduler, it will just wait for the assigned node to become free. Dynamic or scheduling algorithms match tasks to the worker node in the remote scheduler whenever a node becomes idle.

The planning algorithms are disabled by default in Workflowsim. If a planning algorithm is specified, then the scheduling algorithm is disabled, since it forces the Workflow Scheduler to map a task to the resource assigned in the Workflow Planning stage. An example of a planning algorithm included in Workflowsim is Heterogeneous Earliest Finish Time algorithm (HEFT). Examples of scheduling algorithms included in Workflowsim are Min-Min or Max-Min.

3.2 Recreating a cluster in Workflowsim, and multiprogramming the nodes.

Workflowsim places its resources inside datacenters. Each datacenter includes multiple hosts or physical machines. Inside each host, users may create multiple Virtual Machines (VM), and provide them with several CPUs, which in turn contain a set of PUs. As shown in Figure 3.2 (left), the default behavior of a datacenter with multiple VMs in each host is like that of a cloud, where each host plays the role of a cluster with multiples VMs within. As depicted in Figure 3.2 (middle), to generate a multi-core cluster-like environment in Workflowsim, we designed a datacenter with multiple hosts, and a single VM inside each host. In the single-VM-per-host scenario, each host or VM plays the role of a cluster of a node.

In the HBRM validation experiments of Chapter 4, shared-memory multiworkflow tasks are executed in different shared clusters. The maximum DP of each cluster node (or maximum amount of tasks simultaneously executing on clusters' nodes) is 2. As explained in Section 4.1, this amount is chosen since it is enough to accomplish the goal of the experiments, which is to validate the HBRM and show how it operates when it is implemented on shared clusters with multiprogrammed nodes.

To achieve a node sharing with DP=2 in Workflowsim, we have followed the approach shown rightmost part of Figure 3.2. We placed 2 CPUs in each node. Then, we configured each CPU (as explained in Section 3.3), so that only a single task can run on it. Since each node has two

CPUs, and only a single task can run on each CPU, we configured Workflowsim in such a way that the maximum DP of all cluster nodes is 2.

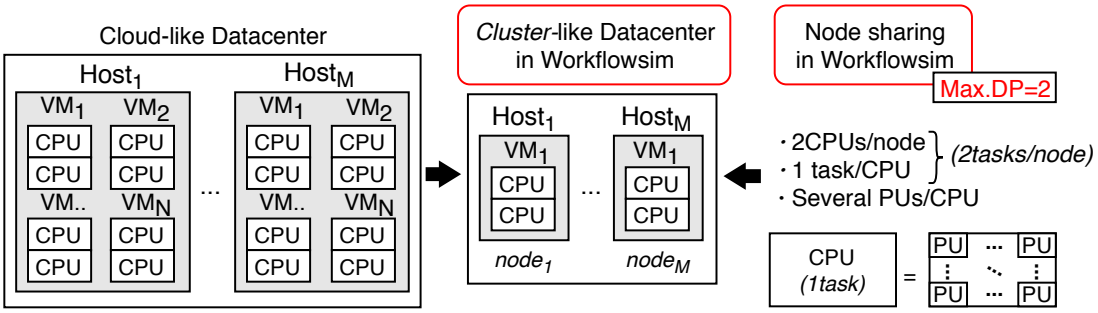


Figure 3.2 Recreation of a cluster-like environment in Workflowsim, and node sharing in Workflowsim. The cluster has been recreated by placing a single VM in each host. The node sharing has been done taking the following approach: placing 2 CPUs in each node, and configuring Workflowsim so that only a single task can run on each CPU (which in turn contains several PUs). Thus, we configured Workflowsim in such a way that the maximum DP of all cluster nodes is 2.

3.3 Modifications done on Workflowsim's main blocks

Workflowsim offers a framework to execute complex workflows in large shared platforms with different scheduling policies. The simulator has a set of scheduling algorithms included by default to schedule resources, such as Min-Min or Max-Min. Users can also implement their own scheduling policies to execute the workflows. In this section, we explain how we implemented the algorithms of the HBRM: SA, Biobackfill, and SAFP.

Each scheduling algorithm allocates tasks to resources and establishes priorities upon different criteria, and has different requirements, such as for instance, makespan predictions. The cluster must be adapted to the requirements of the scheduling algorithms, in order for them to be implemented and function properly. For instance, SA and SAFP algorithms, must be aware of the load running on each cluster node at any time to minimize the makespan slowdown of the tasks sharing nodes.

As we deployed the SA, Biobackfill, and SAFP algorithms in Workflowsim, we found out that a few key considerations required for them to operate were not included. To fulfill the requirements of the HBRM algorithms, and be able to implement them in Workflowsim, some functionalities were added in the simulator. For instance, functionalities that enabled sharing nodes between tasks, or applying the makespan slowdowns yielded.

Below, we list and describe the main modifications done on Workflowsim to adapt it to the SA, Biobackfill, and SAFP algorithms. The modifications are included in two lists. In the first one, we describe the modifications for SA, Biobackfill, as they coincide. In the second list, we describe the modifications of the SAFP algorithm, which is an extension of SA that considers the memory hierarchy to place workflows files.

The modifications done on Workflowsim to adapt it to the requirements of the SA and Biobackfill algorithms of the HBRM for validation are the following ones:

- *Generate CPUs (lists of PUs) and bond each one of them to a different VM.*
  Workflowsim already includes the concept of Virtual Machines (CondorVM objects), and the concept of PUs (Pe objects). Nonetheless, Workflowsim has been designed to allocate tasks to whole machines, rather than to specific PUs within a machine. To implement the HBRM algorithms, we must recreate a scenario where two tasks share a node (DP=2), that is are executed in different PUs of a node. To do so, we generated two CPUs per node. In each CPU, made of multiple PUs, can run one task. To generate unique CPUs, we generated lists of unique PUs (unique Pe objects). Next, we linked to a each node two CPUs (or two PU lists).

  To generate the unique CPUs and link them to cluster nodes, we modified the createVM function. The createVM function originally takes two inputs (int userId, int numVMs), and returns a list of CondorVM objects, each of which has the following parameters: ram, mips, bw, vmName and pesNumber.

  Although each CondorVM object contains the pesNumber parameter, no CPU (list of PUs) is created, nor linked to its corresponding machine. Hence, it wasn't possible for the scheduling algorithms to match tasks with a specifics PUs (Pe objects) of machines. The modification to create a CPU consisted in creating a pesNumber-sized list of Pe objects. Furthermore, a specific amount of mips was assigned to each PU.

- *For each scheduling iteration, generate a list of both running and waiting tasks, readable by the local scheduler (to determine the load of each VM).*
  When considering a function to monitor the load of a VM (cluster node) with a specific ID (VmID), one may think of a getLoad() function. VmID.getLoad() would return the task which is currently running in the VM. Workflowsim monitors the load of the VMs in a different way. There isn't a function that takes the VM identity and returns the running task. Instead, Workflowsim is requires a complete list of tasks, that is, both tasks scheduled in past scheduling iterations and tasks scheduled (or to be scheduled) in the current scheduling iteration. Then, for each task in the list, one must ask whether it has run on a specific machine or not: task.getVmId(), and if so, ask again if the task is still running: task.getCloudletStatus()==running.

  The problem encountered is that Workflowsim scheduling algorithms included by default don't contain a complete list of tasks. They only contain the getCloudletList() list, which includes: all the tasks to be scheduled in the current scheduling iteration, and all the tasks just scheduled in the current scheduling iteration. At the end of each scheduling iteration, getCloudletList() removes all already-scheduled though not-yet-finished tasks. getCloudletList() lacks the tasks that have already been scheduled in past scheduling iterations, and it is essential to have them in order to determine which task is running on each VM. The implemented solution consisted

in backing up past-iteration-scheduled tasks in a list we created (backupCloudletList()), before Workflowsim removes them at the end of each scheduling iteration, and restore the backup when the next scheduling iteration starts. The backup list has to be readable from the file in which the scheduling algorithm is defined. In order for that to happen, proper modifications were conducted in the following files: DatacenterBroker, BaseSchedulingAlgorithm, and NetDatacenterBroker.

- *Create a new status for nodes that are partly-busy (LOADED).*
  There are different labels in Workflowsim to define the different status of the nodes (VMs). Among them one may find the IDLE status, which is given to VMs that are idle (nodes that can host a new task execution), and the BUSY status, which is given to VMs that are idle (nodes that can't host a new task execution). Nonetheless, Workflowsim doesn't consider an intermediate status to define partially-loaded VMs, that is (in our case with DP=2) VMs or nodes that already host one task execution, and can therefore host another task execution. To do so we created the LOADED VM status, and redefined the IDLE VM status and BUSY VM status. Considering that each cluster node has 2 CPUs (each CPU can host one task execution) we assigned VM statuses as follows:

  – IDLE VM status: given to nodes with both CPUs in idle state.
  – *LOADED VM status:* given to nodes with 1 CPU in idle state and 1 CPU in busy state.
  – BUSY VM status: given to nodes with both CPUs in busy state.

- *Apply the makespan slowdowns to tasks running in multiprogrammed nodes (DP=2).*
  The amount of slowdown varies upon the characteristics of the combinations of tasks sharing the same node. To apply the proper amount of slowdown, a slowdown matrix such as that of Table 2.5, must be declared in the scheduling algorithm. However, the slowdown time of 2 tasks sharing a node is not applied over their entire running time, but it is applied over the period of time both tasks are simultaneously run on a node, the overlap time. In Figure 3.3 we depict the concept of overlap time among two tasks sharing a node, each running in PUs of a different CPU. To add the slowdown, the overlap time is multiplied by the amount of slowdown of the Table 2.5. Afterwards, the slowdown is added to the predicted exclusive-mode makespan of each task by using the setCloudletLength() function.

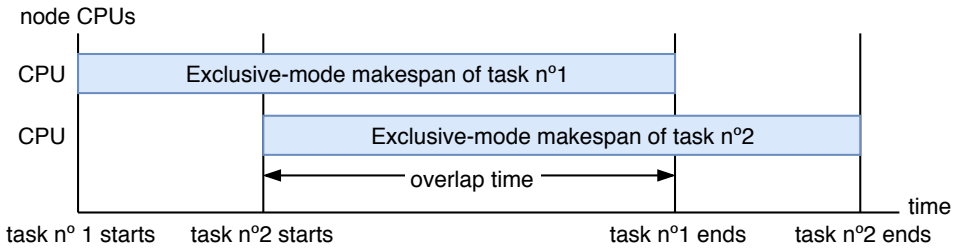


Figure 3.3 Applying the slowdown to the overlap time of 2 tasks running in the same node.

The SAFP algorithm is an extension of the SA algorithm that places the input workflow files on the different memory hierarchy levels of the cluster depending on the size, and on whether the file is shared by more than one task. By default, Workflowsim has two different storage levels: a NFS shared among all nodes of a datacenter, and a local hard disk per node. The SAFP considers a third storage level, the local RAM disk, which we mounted over a portion of the RAM.

The modifications done to Workflowsim to adapt it to the requirements of the HBRM's SAFP algorithm are the following ones:

- *Create a local RAM disk for each node.*
  The SAFP algorithms considers the RAM disk when placing workflows files. The RAM disk is not present in Workflowsim, and we had to include it it. To do so, we created a new class, called RamDiskStorage, and included it in the same Workflowsim package hosting other storage levels such as the local hard disk, defined in the HarddriveStorage class.

- *Attach the RAM disk to the storage list of the cluster.*
  The createDatacenter function includes all the storage levels included on the cluster datacenter. To attach the RAM disk to the datacenter, the createDatacenter function was modified. The RamDiskStorage object was created, given a size, and added to the storage list of the datacenter.

- *Monitoring the contents of the local RAM disks and hard disks of each node.*
  SAFP determines the hierarchy memory level where workflow files must be placed, based on the file size, and whether the it is shared (requested by multiple tasks of a workflow) or non-shared. Before that, SAFP reads the content of the memory levels for two reasons. First, to find out whether the file is already stored in the proper storage unit. Second, to learn whether the target storage unit is full and needs to be freed space. Thus, SAFP must continually monitor the contend of the RAM disks and hard disks. To be able to monitor the local RAM disks and hard disks of all nodes on the cluster, several modifications were made in the replicacatalog class.

Once all the Workflowsim modifications were completed, we were able to implement the SA, Biobackfill, and SAFP algorithms of the HBRM. To validate the correctness of the modifications and ensure that no inaccuracies had been added, we conducted a series of experiments in Section 4.3.

Chapter 4

Validation of the HBRM

In this section we describe the experiments conducted to validate the performance of the proposed HBRM when processing different queues of workflow tasks submitted to clusters, and compare the performance obtained with that of other state of the art resource managing approaches.

We recreate multiple scenarios with different cluster resources where we submit queues of workflow tasks, and process them with the HBRM, addressing all the blocks tasks go through: Application characterization, Multivariate Regression Prediction, and Scheduling Multicriteria (SA, Biobackfill, or SAFP, depending on the case). The same workloads are also processed with policies such as FCFS (First Come First Served), Bestfit, Firstfit, Min-Min and Max-Min. At the end, we compare the different metrics obtained in each scenario and scheduling algorithm: workflow makespans, resource efficiency, and resource usage.

Our goal is to prove that the HBRM processes queues of bioinformatics workflow applications submitted to shared clusters for execution in such a way that the average workflow makespan, efficiency, and resource usage are improved in comparison with other state of the art techniques.

To validate the HBRM, and prove that it indeed improve workflow performance and resource utilization, we generate different scenarios where workloads of bioinformatics workflow applications with parameters and data values are submitted to cluster queues. In each scenario, we implement the HBRM as well as other resource managing approaches in order to process the queue of workflow tasks. Next, we process the queue with the implemented resource managing approaches and analyze the different performance metrics obtained in each case. This section is structured as follows:

- In Section 4.1, we summarize the structure of the different experiments, and describe the common traits of the experiments conducted in the different scenarios, such as the batch calculation or how the dependencies are addressed.
- In Section 4.2, we validate the HBRM in a cluster, which has SLURM as a default RMS. We process a queue of workflow applications in two ways: with the HBRM (implemented before SLURM) using the SA algorithm, and with SLURM's FCFS policy. With these experiments

we prove that the HBRM improves the average workflow makespan, resource efficiency and usage of FCFS.

- As previously explained in Chapter 3, some of the blocks forming the Workflowsim simulator had to be modified to extend their original functionalities, and be able to implement the algorithms of the HBRM. This allowed us to conduct further validation experiments on large clusters and compare the HBRM algorithms with other scheduling policies. In Section 4.3, we validate the correctness of the Workflowsim extensions to implement the SA algorithm. To that end, the exact same experiments done on the cluster in Section 4.2, are repeated in Workflowsim, and makespan differences compared.

- In Section 4.4, we validate the HBRM's BioBackfill algorithm in Workflowsim, and extend the comparison to enhanced policies such as Firstfit and Bestfit. Although we use a simulated cluster, we use as a starting point a moderate amount of nodes. These tests allow us to show that the bioinformatics applications-oriented backfill improves the makespan obtained with other backfill approaches.

- In Section 4.5, we validate the HBRM's SA and SAFP algorithms on large clusters, and compared the makespans obtained with those of Min-Min and Max-Min algorithms. We observed how their performance scaled by conducting validation in a wide range of nodes. To also validate them under heavily loaded conditions, we selected multiple instances of three well-known large workflows of different topology and tasks characteristics: Epigenomics, Montage, and Sipht.

4.1 Introduction to the validation experiments

Our objective is to validate the HBRM and all its blocks: Application characterization, Multivariate Regression Prediction, and Scheduling Multicriteria. The validation is done with different HBRM algorithms (SA, Biobackfill, and SAFP) in different scenarios (workloads and resources). Furthermore, the performance of the HBRM algorithms is compared with those of other state of the art policies (Bestfit, Firstfit, Min-Min or Max-Min), as well as FCFS. However, regardless of the different workloads, clusters or policies, all the validation experiments have some traits in common. In this section, we introduce the experiments and review their common considerations. To do so, we will follow Figure 4.1 from top to bottom, which depicts the general layout of the different validation experiments. The workload included in Figure 4.1 has been taken from Figure 4.6.

For all the validation experiments, we consider all cluster nodes to be empty before the submission of the workflows starts. All the experiments start by considering a series of users dynamically submitting a series of workflows formed by different bioinformatics applications, with given parameter values, and data, to a cluster. The performance criteria considered for the

users is the assumed to be the minimization of the makespan of their workflows. To minimize the monetary cost of the executions, the same approach would apply, as the cost can be easily derived form the makespan.

In all cases users don't know the resources their applications need to minimize the makespan of their workflows, and don't provide resource descriptions. On the other hand, the cluster administrator focuses on maximizing the resource utilization.

For all the validation experiments of this work, we chose a maximum degree of multiprogramming of 2 (DP=2) in all cluster nodes. We considered this amount since the it allows us to show the benefits of multiprogramming nodes, rather than optimizing the maximum amount of applications the node is multiprogrammed with. However, the proposal is also valid for higher degrees of multiprogramming. All cluster nodes can have three statuses: idle, loaded or busy. Idle nodes are those that are not hosting any execution, that is have all PUs idle. Loaded nodes are those that are hosting the execution of one task, that is have some PUs idle and some others busy. Busy nodes are those that are hosting the execution of two tasks, that is have all PUs busy.

The dynamically-submitted queue of tasks is received by the HBRM, and goes through the Multivariate Regression Predictor. For each workflow task submission, the predictor searches in the historical database for similar past executions run on the cluster. That is, executions of the same application and performance criteria (ρ=makespan), with similar parameters and data. If found, the Multivariate Regression Predictor of the HBRM generates multiple makespan and cost predictions with different combinations of resources, in all nodes with a wide range of PUs. The multiple predictions help determine the proper resources regardless of the resource availability. It also allows for multiple performance metrics to be calculated, such as the efficiency or speed up of each task in each node, as depicted in Figure 2.8. With these metrics, the amount of PUs each task needs to reach the scalability threshold or maximum speed up, $numPUs_{MaxSpeed}$, can be calculated. Also can be calculated the makespan penalties obtained when running with an amount of PUs inferior to $\text{numPUs}_{\text{MaxSpeed}}$, that is Section 4.2. Generating multiple predictions for each allows to have enough information to balance the trade-offs among users seeking the minimization of their workflow makespans, as well as the trade-off between each user and the platform administrator, who seeks to maximize the overall resource utilization.

The rate at which users dynamically submit their workflows, as well as their resource requirements or expected duration, often surpasses the capacity of shared clusters to execute them all, and jobs are prompted to wait in the queue. In some cases, the amount and requirements of submissions overwhelms by far the capacity of the resources of the cluster, reaching a point where the waiting times of some jobs soar, growing unreasonably long in comparison with for instance, their makespan.

To prevent that from happening, the cluster submission queue is divided in multiple processing batches. Those batches that can be attended within reasonable time are granted admission, and the rest are filtered out. We determined the batch based on the two most intensively used

resources of the considered bioinformatics applications: the amount of PUs and the memory capacity of the nodes. First, we accounted for the aggregated number of PUs all (x) tasks in the queue need to run with maximum speed: $\sum_{ID=1}^{x} numPU_{MaxSpeed}$, and the aggregated memory capacity of the nodes they require: $\sum_{ID=1}^{x} Memory$.

Second, we set the size of each batch based on the aggregated number of PUs and memory available of all (n) nodes of the cluster: $PUs_{Cluster} = \sum_{Node=1}^{n} PUs_{Node}, Mem_{Cluster} = \sum_{Node=1}^{n} Mem_{Node}$. And introduced in each batch as many submitted applications as can be executed running with $\text{numPUs}_{\text{MaxSpeed}}$ with: $PUs_{Cluster}$ and $Mem_{Cluster}$. In all validation scenarios multiple batches have been selected, in order to generate enough workload to fill the cluster and expose the benefits of the HBRM. In Figure 4.1, the admitted workflow applications are represented in different colors, whereas the rejected ones are in white color.

It must be pointed out that the different HBRM algorithms considers the dependencies of the multiple workflows submitted in the queue. However, in each scheduling iteration, the HBRM algorithms only schedules lists of ready workflow tasks. That is, tasks whose dependencies have been solved. The list of ready tasks can be seen in Figure 4.1. Based on makespan predictions and slowdown information, the HBRM-chosen algorithm: SA, Biobackfill or SAFP, arranges the queue of admitted ready workflow applications. The HBRM adjusts the amount of resources allocated, such as PUs, to the actual needs of applications, minimizing the makespan of the applications running in the multiprogrammed nodes. Furthermore, when deciding which combinations of applications are to share the same nodes, it accounts for the resource usage applications carry out, in such a way that the makespan slowdown of applications in multiprogrammed nodes is minimized.

4.2 Multiworkflow HBRM validation on clusters with the SA algorithm

In this section, we validate the HBRM in a shared cluster with heterogeneous nodes featuring SLURM as default RMS. To do so, we process a workload on the cluster, with resources being managed in two different ways: with the HBRM (SA algorithm) alongside SLURM, and solely with SLURM's FCFS. Once the workload is processed, we analyze the different performance metrics obtained with both methods: average workflow makespan, cluster resource usage, and efficiency. The goal of these experiments is to prove not only that the HBRM is compatible to be implemented alongside SLURM, but also that it can improve the performance of bioinformatics workflows on shared clusters obtained with SLURM.

The efficiency of applications in exclusive mode has been calculated as the speed up divided by the number of processors, as in Equation 4.1. Similarly, the efficiency of multiple applications in shared mode, i.e. App_A and App_B executed simultaneously, has been calculated as in Equation

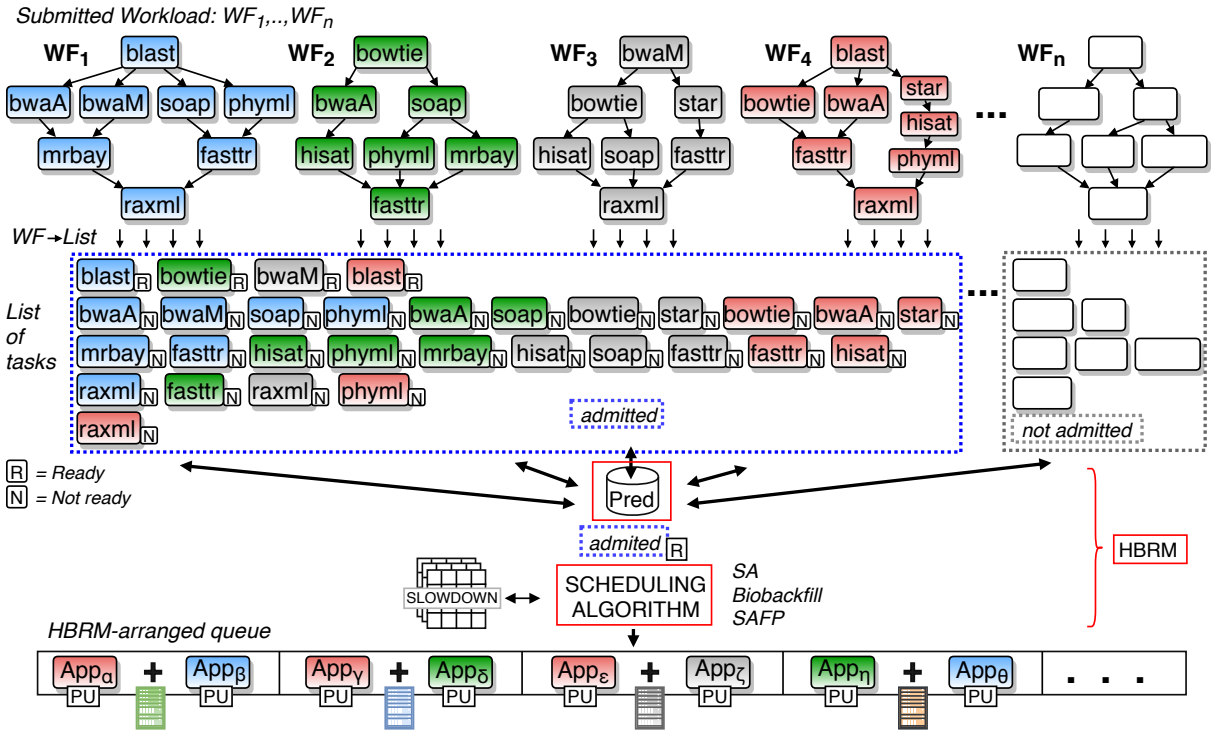


Figure 4.1 General layout of the different HBRM validation experiments. From top to bottom: (i) workflows dynamically submitted by users, (ii) list of ready applications (admitted ones within blue-dashed rectangle), (iii) scheduling algorithm allocating based on predictions and slowdowns, (iv) priority-sorted list of applications with resources allocated. Applications within the same rectangle are scheduled for same-node execution.

5. The resource usage has been calculated as the ratio between the aggregated number of PUs used by applications running on a node, and the number of PUs of the node.

$$Efficiency(App)^{nPU}_{Exclusive} = \frac{Makespan(App^{1PU})}{Makespan(App^{nPU}) * n} \tag{4.1}$$

$$Efficiency(App_A^{nPU}, App_B^{mPU})_{Shared} = \frac{Makespan(App_A^{1PU})_{Excl.} + Makespan(App_B^{1PU})_{Excl.}}{Makespan(App_A^{nPU}, App_B^{mPU})_{Shared} * (n+m)} \tag{5}$$

4.2.1 Workload

To conduct the experiments, we generated a workload formed by two multiworkflows. MWF_A and MWF_B, which are executed by separate in cases A and B, respectively. The workload is made up of already-characterized workflow applications of the Table 2.1.

The MWF_A, executed in the case A, contains 4 workflows of 30 applications: WF_1, .., WF_4. The MWF_B, executed in the case B, contains 6 workflows of 52 applications: WF_1, .., WF_6. The same applications are submitted with different parameters and data in each workflow. Thus:

$Raxml_{WF1}$, $Raxml_{WF2}$, .., $Raxml_{WF6}$ have different makespans. The structure of the 6 workflows processed in shown in Figure 4.2.

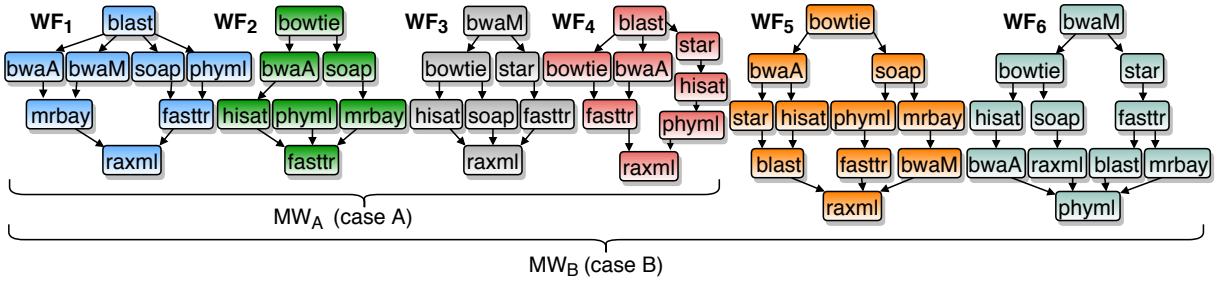


Figure 4.2 Workload generated to validate the HBRM with the SA algorithm on clusters. It is composed of two multiworkflows: MWF_A (30 applications), and MWF_B (52 applications). The same applications have different makespans in each workflow.

4.2.2 Resources

The main specifications of the cluster used to validate the HBRM with the SA algorithm are described in Table 4.1. The cluster is formed by four nodes with heterogeneous characteristics and large memory capacities. Its default RMS is SLURM, version 16.05.9.

Table 4.1 Main specifications of the cluster used.

Node Architecture	RAM	PUs	Local Disk	Clock rate
AMD IO-6376	128GB DDR3	64	250 GB	2.3GHz
AMD IO-6376	128GB DDR3	64	250 GB	2.3GHz
Intel Xeon E5-4620	128GB DDR3	24	250 GB	2.2GHz
Intel Xeon E5-2620	64GB DDR3	24	250 GB	2.1GHz
Aggregated	448 GB	176		

4.2.3 Overview of the experiments and analysis of the results

To validate the HBRM on clusters with the SA algorithm we process a workload twice. First, with resources being managed by the HBRM with the SA algorithm, placed alongside SLURM. Secondly, with resources being managed by SLURM's FCFS. At the end, we compare the average workflow makespans, resource efficiencies and resource usages obtained with both approaches. The layout of the experiments is depicted in Figure 4.3.

The experiments to validate the HBRM on clusters with the SA algorithm start with the dynamic submission of a multiworkflow, for both cases A and B. The queue is received by the HBRM, and all tasks within go through the Multivariate Regression Predictor. For each submitted task, the Multivariate Regression Prediction, queries the database, looking for similar past executions, as depicted in Figure 2.4. For each submitted task with similar past executions

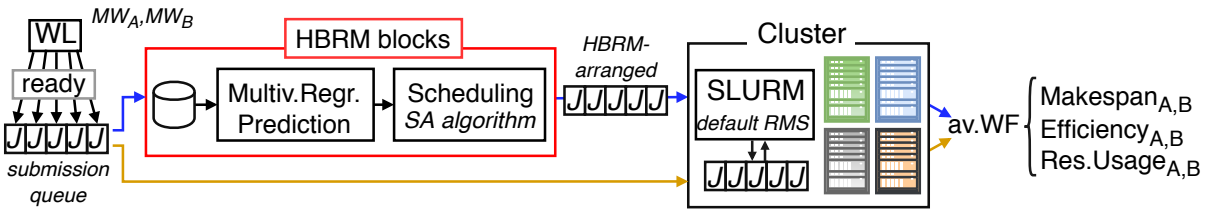


Figure 4.3 Layout of the experiments to validate the HBRM with the SA algorithm on clusters.

found in the database, the predictor generates makespan predictions with different combinations of resources: node architectures and range of PUs.

The cluster queue is then divided into multiple admission batches. As explained in Section 4.1, Batches are calculated based on the predominant resources used by the considered bioinformatics applications: the PUs and the memory. For the present experiments we chose 2 and 4 workflows, since it is enough to explain the proposal. In the following lines we explain with the given workload and cluster how the batch calculation has been done.

The cluster, described in Table 4.1, has 176 PUs and 448GB of memory capacity. Applications are executed with $\text{numPUs}_{\text{MaxSpeed}}$. As the memory capacity of the clusters is considerable large (448G), and poorly managed by the considered bioinformatics applications, the amount of PUs is the resource that limits the amount of bioinformatics applications that can be run simultaneously with $\text{numPUs}_{\text{MaxSpeed}}$ on the cluster. Thus, we use the PUs as the resource to explain the amount of batches. A batch will be formed by as many applications as can be executed with $\text{numPUs}_{\text{MaxSpeed}}$ with the 176 PUs. Table 4.2 shows the average $\text{numPUs}_{\text{MaxSpeed}}$ of each application in all 6 workflows, obtained from the predictor.

Table 4.2 Average predicted $\text{numPUs}_{\text{MaxSpeed}}$ of each application in all workflows, obtained from the predictions generated by the Multivariate Regression Predictor.

Blast	**BwaM**	**Bowtie**	**BwaA**	**Hisat**	**Star**	**Soap**	**Phyml**	**Mrbayes**	**Fasttree**	**Raxml**	**Aggregate**
16	16	16	16	8	8	8	16	16	8	8	**136**

From Table 4.2, it can be calculated that the 30 applications of the WF_1, .., WF_4 account for an aggregated amount of 384 PUs for max speed, and the 52 applications of WF_1, .., WF_6 account for 656 PUs for max speed. As 176 x 2 = 352 (384), and 176 x 4 = 704 (656), it can bee stated that approximately 2 batches (4 workflows of case A), and 4 batches (6 workflows of case B) have been selected for the experiments.

The HBRM considers for scheduling a list of workflow applications. Thus, workflows are broken down into a list of tasks. From the 4 workflows of case A, 30 applications are obtained. From the 6 workflows of case B, 52 applications are obtained.

The admitted lists of tasks, go through the Multivariate Regression Prediction Model once more. For each admitted task, the predictor queries in the database as depicted in Figure 2.4, looking for similar past executions. If found, generates the makespan predictions with different

resources.

In Table 4.3, we include a summary of the makespan predictions of the admitted list of applications, generated by the Multivariate Regression Prediction. To provide a clearer explanation, in Table 4.3 we only include the average predicted makespan (in seconds) of each application in all cluster nodes with $\text{numPUs}_{\text{MaxSpeed}}$, for cases A and B.

Table 4.3 Summary of the performance predictions generated by the Multivariate Regression Predictor: all-node average predicted makespans (in seconds) of the admitted workflow applications with $\text{numPUs}_{\text{MaxSpeed}}$, for cases A (4 workflows, 30 applications) and B (6 workflows, 52 applications).

		Admitted	**Blast**	**BwaM**	**Bowtie**	**BwaA**	**Hisat**	**Star**	**Soap**	**Phyml**	**Mrbayes**	**Fasttree**	**Raxml**
case A	**case B**	**WF_1**	3640	3313	-	4136	-	-	2921	1622	3079	3000	2767
		WF_2	-	-	2934	5688	864	-	4021	3068	3510	1985	-
		WF_3	-	4576	3681	-	1220	1207	5362	-	-	3265	3443
		WF_4	7021	-	5420	8346	1966	1572	-	3753	-	6508	4099
		WF_5	4787	4015	4255	5102	2418	6791	4534	3970	3391	3056	3197
		WF_6	3040	6254	4500	4167	3567	5615	4129	3826	2988	3719	2699
		MRE	8.1%	8.6%	9.2%	10.7%	8.4%	10 %	9 %	9.3%	10.5%	9.4%	9.6%

For cases A and B, the scheduling algorithm receives the makespan predictions summarized in Table 4.3, and the slowdown information summarized in Table 2.5. The makespan predictions reveal the amount of resources for fast execution with an adjusted amount of PUs. Compatibility slowdown information is employed to multiprogram the nodes. This way, applications can be combined for same-node execution in such a way that slowdown is reduced as much as possible.

As mentioned before, cases A and B are processed by separate on the cluster in two different ways: by using the HBRM (SA algorithm) alongside SLURM, and by using only SLURM.

Based on applications' performance information in both shared and exclusive mode, the HBRM's SA algorithm outputs a list of applications and resources sorted by priority, which is sent to the cluster's default RMS, SLURM. SLURM receives the HBRM-managed queue, providing implicit resource scheduling information that is based on a thorough study and knowledge of applications' characteristics. That is, receives a queue in which, as shown in Figure 4.1, bottom, each task is allocated an amount of PUs matching their resource needs. Also tasks are prioritized in so that the different combinations of tasks share nodes by accounting for the slowdown. Assisted by the information sent by the HBRM, and based on its own implemented policies, SLURM schedules the queue. A scheduling policy of SLURM compatible with the HBRM implementation is the FCFS, as it doesn't alter the input and allocates tasks to the cluster as they arrive.

Results obtained after processing the 4 workflows of case A, with the two processing approaches, are provided in Table 4.4. Three performance metrics: makespan, efficiency and resource usage, have been calculated. For the given case, employing the HBRM (characterization, prediction, scheduling), improves average workflow makespan by up to 28%, average workflow

efficiency by up to 75%, and average resource usage of workflows by up to 101%, compared to solely using SLURM.

Table 4.4 Workflow makespans (in seconds), efficiencies and resource usages obtained when processing the 4 workflows of case A, with HBRM (SA algorithm) alongside SLURM, and only SLURM.

	Makespans case A			**Efficiencies case A**			**Resource Usages A**		
	HBRM (SA)	**SLURM**	**Improv.**	**HBRM (SA)**	**SLURM**	**Improv.**	**HBRM (SA)**	**SLURM**	**Improv.**
WF_1	17959	23057	22%	0.43	0.25	73%	99%	49%	102%
WF_2	18243	23589	23%	0.52	0.30	75%	88%	46%	91%
WF_3	18175	25963	30%	0.52	0.28	85%	80%	44%	82%
WF_4	17959	27958	36%	0.52	0.31	68%	87%	36%	138%
Av.	*18084*	*25141*	*28%*	*0.50*	*0.29*	*75%*	*88%*	*43%*	*101%*

Once case A finishes, the 6 workflows of case B are processed in the same both ways. Case-B results are provided in Table 4.5. For the given case, the HBRM improves average workflow makespan by 35%, average workflow efficiency by 83%, and average resource usage of workflows by 95%, compared with SLURM.

Table 4.5 Workflow makespans (in seconds), efficiencies and resource usages obtained when processing the 6 workflows of case B, with the HBRM (SA algorithm) alongside SLURM, and only SLURM.

	Makespans case B			**Efficiencies case B**			**Resource Usages B**		
	HBRM (SA)	**SLURM**	**Improv.**	**HBRM (SA)**	**SLURM**	**Improv.**	**HBRM (SA)**	**SLURM**	**Improv.**
WF_1	24762	39005	37%	0.64	0.31	108%	99.5%	52%	91%
WF_2	26401	37589	30%	0.83	0.45	84%	92%	50%	84%
WF_3	27125	40212	33%	0.61	0.33	87%	82%	47%	76%
WF_4	27603	39616	30%	0.47	0.33	41%	89%	40%	126%
WF_5	31011	52546	41%	0.78	0.43	81%	95%	49%	94%
WF_6	29611	50223	41%	0.77	0.39	94%	99.8%	49%	102%
Av.	*27752*	*43198*	*35%*	*0.68*	*0.37*	*83%*	*93%*	*48%*	*95.7%*

The average makespan of the HBRM increases from case A (18084 seconds) to case B (27752 seconds) due to the saturation of the resources. However, HBRM's efficiency improves (from 0.5 in case A to 0.68 in case B), and so does HBRM's resource usage (from 88% in case A to 93% in case B). As it can be seen, the resource usage starts approaching its maximum as more and more applications are scheduled on the same amount of resources. This situation may overwhelm the resource capacity, increasing the makespan obtained. On the other hand, new applications included for the case B may generate new combinations of applications leading to improved slowdown and efficiency.

Increasing the workload while maintaining the same amount of resources can have repercussions on overall makespans, as the limited capacity of the cluster gets closer to be fully occupied. Results such as the obtained ones with different multiworkflows can be employed observe how the size of the batches containing the workload admitted affects the makespan improvements.

With this information, a new approach to determine the proper batch size for a given cluster capacity, can be developed.

4.3 Validation of the modifications in Workflowsim to implement HBRM algorithms

In order to implement the SA and SAFP algorithms in Workflowsim, some of its blocks had to be modified. When modifying a tool, biases may be introduced, compromising the accuracy of the simulation tool. results versus those obtained with real-environment tests. To prevent that from happening, and ensure the correctness of the modifications, it is necessary to review how the tool operates with the modifications done. In this section we validate the correctness of the modifications done on Workflowsim in order to be able to implement the SA and SAFP algorithms.

To do so, we conduct in Workflowsim the experiments previously conducted on the cluster (described in Section 4.2). We simulate in Workflowsim the same cluster resources, and process the same workload with the same policies (SA and FCFS). Finally, to assess the accuracy of Workflowsim with the modifications, we compare the makespans previously obtained in the real cluster, with those obtained in Workflowsim.

4.3.1 Workload and Resources

The experiments to validate the modifications introduced in Workflowsim are done by processing the same workload than that of the real-cluster experiments of Section 4.2. Thus, a workload of two multiforklows, is processed. In the case A, MWF_A, containing 4 workflows (WF_1, .., WF_4) and 30 applications, is processed. In the case B, MWF_B, formed by 6 workflows (WF_1, .., WF_6) and 52 applications, is processed. The six workflows, depicted in Figure 4.4, are made up of already-characterized applications extracted from Table 2.1. The same applications have different parameters and data in each workflows, and thus different makespans.

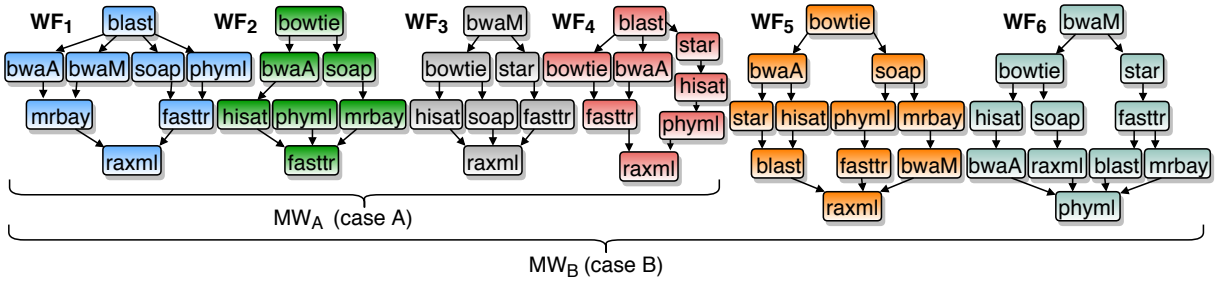


Figure 4.4 Workflows processed to validate the tweaks added in Workflowsim.

The resource specifications of the simulated cluster used to validate the Workflowsim modifications have been adjusted to be as similar as possible to those of the real-cluster experiments,

described in Table 4.1. The resulting specifications of the simulated cluster are are exposed in Table 4.6.

Table 4.6 Specifications of the cluster.

	Simulator Specifications
Nodes p/Cluster	4
Processors p/Node	24; 24; 64; 64 (in 2 CPUs)
Processors Freq.	6000 MIPS
RAM p/Node	96 GB
Disk capacity p/Node	1 TB
Net latency	0.2 ms
Internal latency	0.05 ms

4.3.2 Overview of the experiments and analysis of the results

The goal of the experiments is to ensure that the modifications included in Workflowsim to implement the SA algorithm have introduced no errors in the simulator. To do so, we repeat in Workflowsim the same experiments previously conducted on the cluster, and evaluate the differences between the makespans obtained in the real and simulated cluster

In all Workflowsim experiments, the simulator has been configured so that the maximum DP of the nodes is 2. By default, Workflowsim is configured for allocating single tasks to a fixed amount of PUs (Pe objects), and not PUs within a node. To allocate two tasks within a node, and get to share node resources among 2 tasks, we have followed the approach shown the rightmost part of Figure 3.2. We placed 2 CPUs in each node. Then, we configured the CPUs (as explained in Section 3.3), so that only a single task can on each CPU. Finally, we linked a pair of CPUs to each node.

Workflowsim deals with workflows that are defined in XML files. To process the workload of Figure 4.4 we generated two XML files: in the first one we defined the MWF_A, and in the second one, the MWF_B. Next, we processed each XML file with the SA and with the FCFS algorithms. In Figure 4.5 we can see a sample of the XML file in which the MWF_A was defined. The sample displays how the Bwa-mem task of WF_1, and the Phyml task of WF_1, are described. Within it, we can see the input files(link="input"), and output files (link="output") each task, as well as the file names (uses file="filename"), the size (in bytes), and the makespan predicted with the Multivariate Regression Predictor of the HBRM (predictedMakespan, in seconds).

Workflowsim is a simulator that by default is configured to execute applications with a unique amount of PUs. In fact, in all the XMLs included in the Workflowsim package, the number of PUs of each *job id* is not specified, and each workflow task is described with a name, execution time and a set of input and output files.

In order to fit the characteristics of Workflowsim and be able to test the HBRM, an approach that seeks minimizing the workflow makespans and maximizing resource usage with

multiprogrammed nodes, the *predictedMakespan* that we included in the XML files, is the one the predictor estimated with $\text{numPUs}_{\text{MaxSpeed}}$ (shown in Table 4.3). To simulate the makespan slowdown of the multiprogrammed cluster nodes (DP=2), we have included in the algorithm the slowdown matrix of Table 2.5. The slowdown time has been applied onto the overlap time, as depicted in Figure 3.3.

```
...
<job id="ID00001" name="Bwamem_WF1" predictedMakespan="4136" cpus="1" >
    <uses file="out.sam" link="output" size="39270629769"/>
    <uses file="hgA1f_30G.fq" link="input" size="32212254569"/>
    <uses file="hs_v37.fa.amb" link="input" size="8574"/>
    <uses file="hs_v37.fa.ann" link="input" size="4028"/>
    <uses file="hs_v37.fa.bwt" link="input" size="3137161344"/>
    <uses file="hs_v37.fa.pac" link="input" size="784290318"/>
    <uses file="hs_v37.fa.sa" link="input" size="1568580688"/>
</job>
...
<job id="ID00004" name="Phyml_WF1" predictedMakespan="1622" cpus="1">
    <uses file="out.sam" link="output" size="24568"/>
    <uses file="all_seq.phy" link="input" size="2555450"/
</job>
...
```

Figure 4.5 Sample of the XML in which the MW_A was defined. It contains the descriptions of Bwa-mem of WF_1, and Phyml application of WF_1. For each application the XML requires: input and output file names, sizes (in bytes), and the makespan predicted with the Multivariate Regression Predictor of the HBRM (predictedMakespan, in seconds).

As in the cluster experiments, the makespan predictions are assumed to available before execution. Thus, we included them in the corresponding XML files. The makespans obtained after processing the workload with Workflowsim are included in the Table 4.7 (Workflowsim columns). Also, we included the makespans previously obtained on the cluster (Cluster columns), in order to facilitate the comparison.

Table 4.7 Makespans (in seconds) of the experiments carried out to validate modifications done on Workflowsim. The same workflows are processed recreating the same conditions: workloads, scheduling policy and resources.

	Makespans case A			**Makespans case B**		
	HBRM (SA) *Workflowsim*	**HBRM (SA)** *Cluster*	**Difference** *Cluster-Wfsim*	**HBRM (SA)** *Workflowsim*	**HBRM (SA)** *Cluster*	**Difference** *Cluster-Wfsim*
WF_1	19754	17959	10%	26991	24762	9%
WF_2	18790	18243	3%	27985	26401	6%
WF_3	19083	18175	5%	29837	27125	10%
WF_4	19395	17959	8%	29673	27603	7.5%
WF_5	-	-	-	33147	31011	6.9%
WF_6	-	-	-	32139	29611	8.5%
Av.	*19256*	*18084*	*6.5%*	*29963*	*27752*	*8%*

By comparing the real and the simulated makespan results, we can determine the deviation obtained with the modified version of Workflowsim. As it can be seen, little makespan difference is obtained when simulating the processing of MWF_A and MWF_B, in comparison with doing so in the real cluster. In case A, the average workflow makespan obtained with Workflowsim differs by 6.5% with respect to the workflow average makespan obtained on the cluster. In case B, the average workflow makespan with Workflowsim differs 8% with respect to that of the cluster.

The little variation yielded when replicating the tests in simulated environment, 6.5% and 8% for cases A and B, determine the modifications undertaken in Workflowsim in order to adapt it to the implement the SA algorithm, have been correctly done. At this point, the behavior of the extended Workflowsim has been validated, and it can be considered ready to undergo further testing with more-complex conditions. Its resilience allows us to pave the way to conduct further tests with greater workloads and resources.

Out of these comparisons the pre-processing time can be obtained too: 1172 seconds for case A, and 2210 seconds for case B. As it can be observed, the amount of time spent in this phase depends on the number and complexity of the input applications, and the number of available resources. In a context with many applications and limited amount of resources, finding a scheduling solution becomes more complex and thus more time-consuming than in contexts with many applications but scarce resources.

4.4 Multiworkflow HBRM validation on clusters with the Biobackfill algorithm

In this section, we validate the HBRM in a shared cluster using Workflowsim simulator. To do so, we process a workload on the cluster, with resources being managed in three different ways: with the HBRM using the Biobackfill algorithm, and with the Firstfit and Bestfit algorithms. Once

the workload is processed, we analyze the average workflow makespan, cluster resource usage, and efficiency. The goal of these experiments is to prove that the HBRM with the Biobackfill algorithm improves the average workflow makespan obtained with Firstfit and Bestfit. The same applications have different parameters and data in each workflow, and thus different makespans.

4.4.1 Workload

To conduct the experiments, we generated a multiworkflow composed of 4 workflows: WF_1, WF_2, WF_3 and WF_4, and 44 applications. The structure and applications of the multiworkflow can be seen in Figure 4.6. The workload is made up of already-characterized workflow applications of the Table 2.1.

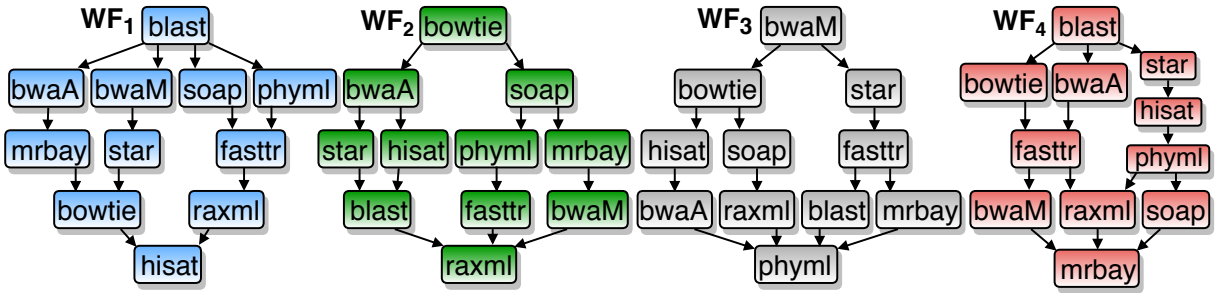


Figure 4.6 Workload generated to validate the HBRM with the Biobackfill algorithm on clusters. It is composed a multiworkflow of 44 applications. The same applications have different makespans in each workflow.

4.4.2 Resources

The first round of HBRM validation experiments in Workflowsim have been carried out in a cluster of similar specifications than those of the cluster of Table 4.1, used for the experiments of Section 4.2.

Thus, to validate the HBRM with the Biobackfill policy, we recreated in Workflowsim a cluster with 4 nodes. The main specifications of the cluster are displayed in Table 4.8

Table 4.8 Specifications of the simulated cluster.

	Simulator Specs
Nodes p/Cluster	4
Processors p/Node	64; 64; 24; 24 (2CPUs)
Processors Freq.	6000 MIPS
RAM p/Node	96 GB
Disk capacity p/Node	1 TB
Net latency	0.2 ms
Internal latency	0.05 ms

4.4.3 Overview of the experiments and analysis of the results

To validate the HBRM on clusters with the Biobackfill algorithm we processed a workload in three different manners. Firstly, with resources being managed by the HBRM with the Biobackfill algorithm. Secondly and Thirdly, with resources being managed by Bestfit and Firstfit algorithms, respectively. At the end, we compare the average workflow makespans obtained with the three different approaches. The layout of the experiments is depicted in Figure 4.3.

As explained with more detail in Section 4.3.2, all the validation experiments in Workflowsim have been conducted with DP=2. To allocate two different tasks in the same node, we have followed the approach shown rightmost part of Figure 3.2. Each PU has been equipped with 2 CPUs, and in each CPU has been placed one task. To simulate the makespan slowdown of the multiprogrammed cluster nodes (DP=2), we have included in the algorithm the slowdown matrix of Table 2.5. The slowdown time has been applied onto the overlap time, as depicted in Figure 3.3.

As also stated in Section 4.3.2, in the XML files in which the workflow tasks are defined, each task has been given the makespan predicted By the Multivariate Regression Predictor with $\text{numPUs}_{\text{MaxSpeed}}$ (shown in Table 4.9).

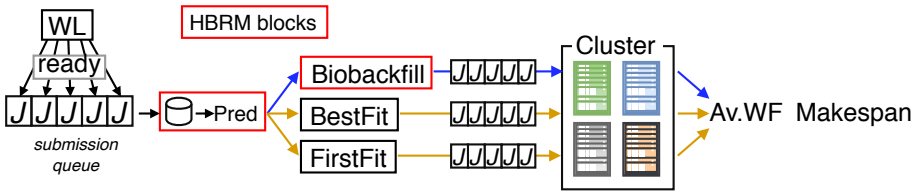


Figure 4.7 Layout of the HBRM validation experiments with the Biobackfill algorithm conducted on the cluster.

The experiments to validate the HBRM on clusters with the Biobackfill algorithm start with the dynamic submission of a queue of tasks belonging to different clusters. The queue is received by the HBRM, and all submissions within go through the Multivariate Regression Predictor, which queries the database and generates makespan predictions for each submission with different resources. The cluster queue is then divided into multiple admission batches, as explained in Section 4.1. Batches calculated based on the predominant resources used by the considered bioinformatics applications: the PUs and the memory. For the present experiments, we selected enough batches to explain the proposal. For the present case, we selected 3 batches. From the batches, 4 workflows and a list of 44 applications were obtained. The list of applications contains both ready and non ready tasks. In each scheduling iteration, the algorithms schedule a list of workflow applications whose dependencies are solved.

In Table 4.9, we include a summary of the makespan predictions of the admitted list of admitted workflow applications generated by the Multivariate Regression Predictor. To summarize the information, only include the average predicted makespan (in seconds) of each application in all cluster nodes with $\text{numPUs}_{\text{MaxSpeed}}$.

Table 4.9 Summary of the performance predictions generated by the Multivariate Regression Predictor of the HBRM: all-node average predicted makespans (in seconds) of the admitted workflow applications with $\text{numPUs}_{\text{MaxSpeed}}$.

Admit.	Blast	BwaM	Bowtie	BwaA	Hisat	Star	Soap	Phyml	Mrbay.	Fasttree	Raxml
$\mathbf{WF_1}$	4851	4136	3464	4391	2001	5527	5183	3719	3264	3262	3032
$\mathbf{WF_2}$	3760	4688	4866	4947	4145	4338	3779	3750	3890	3266	3686
$\mathbf{WF_3}$	4814	4547	4518	4099	4749	3359	4926	2368	3551	4554	3524
$\mathbf{WF_4}$	4525	5068	4871	4239	4136	4818	3445	3272	3510	4462	3309

Based on applications' performance information in both shared and exclusive mode, the HBRM's Backfill algorithm outputs a list of applications and resources sorted by priority, as shown in the bottom part of Figure 4.1.

Results of Table 4.10 show how the proposed Biobackfill scheduler, by including the slowdown (DP >1) as parameters to choose which candidates are backfilled, can achieve 10% workflow makespan reduction compared to Firstfit, and 7.3% compared to Bestfit. Hence, the HBRM's SA algorithm improves the average workflow makespans of both state-of-the art backfill policies by 8.55%, on average.

Table 4.10 Makespans in seconds obtained after processing the workflows with different backfill policies. Average improvement of the Biobackfill versus Firstfit and Bestfit.

	HBRM (Biobackfill)	Firstfit	Bestfit	Improvement vs Av. (Firstfit,Bestfit)
$\mathbf{WF_1}$	27899	29229	30248	6.19%
$\mathbf{WF_2}$	30584	32407	38864	3.33%
$\mathbf{WF_3}$	29211	33642	32232	11.31%
$\mathbf{WF_4}$	28388	33642	31895	13.37%
Av.	*29020*	*32230*	*31060*	*8.55%*

4.5 Multiworkflow HBRM validation on large clusters with SA and SAFP algorithms

In this section we validate the HBRM with the SA and SAFP algorithms on large clusters with Workflowsim, and compare them with state of the art Min-Min and Max-Min algorithms. To do so, we generate different multiworkflows composed of multiple instances of Epigenomics, Montage and Sipht workflows, and process them in different clusters using Workflowsim simulator. Once finished the processing, we compare the multiworkflow makespan obtained with all approaches, that is, the time it takes to process the whole queue. The mentioned workflows are included in Workflowsim's default package.

The first goal of the experiments is to prove that HBRM with the SA and SAFP algorithms can improve the makespan obtained with Min-Min and Max-Min scheduling policies. The second goal is to prove that the makespan improvements of the HBRM are also achieved on large clusters with a wide range of PUs, different workload topologies, and heavily-loaded conditions.

4.5.1 Workload

To validate the HBRM with the SA and SAFP algorithms in large environments we chose a series of complex well-known public workflows with different topologies, tasks characteristics and file sizes:

- 3 Epigenomics workflows with 24, 100, and 997 tasks (E_{24}, E_{100}, E_{997}).
- 3 Montage workflows with 25, 100, and 1000 tasks (M_{25}, M_{100}, M_{1000}).
- 3 Sipht workflows with 30, 100, and 1000 tasks (S_{30}, S_{100}, S_{1000}).

Each of the listed workflows is in the public domain, and has been characterized in previous research [8]. The main characteristics of the 9 workflows chosen for the experiments are summarized in Table 4.12.

Table 4.11 Characteristics of the 9 rkflows chosen to generate the multiworkflows to validate the HBRM with the SA and SAFP algorithms on large clusters.

Workflow	Epigenomics			Montage			Sipht		
Number of tasks	24	100	997	25	100	1000	30	100	1000
Total size of files (GB)	8.3	124.1	1223.7	0.3	1.4	14.0	0.7	2.0	20.5
Av. size of files (MB)	249.9	858.8	842.7	3.7	3.3	3.2	0.4	0.4	0.3

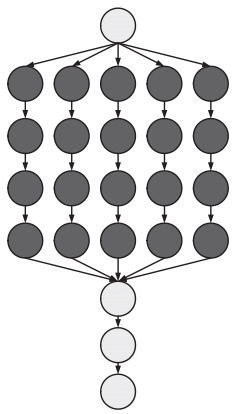
The different workflows selected and amounts of applications forming them allow us to test the scheduling policies under different workloads with different amount of applications, different characteristics and topologies. To generate even-bigger workloads, and see how the HBRM algorithms respond, we generated 9 multiworkflows composed of different instances of the 9 workflows mentioned in Table 4.12.

The structures of the Epigenomics workflow with 24 tasks (E_{24}), Montage workflow with 25 tasks (M_{25}), and Sipht workflow with 30 tasks (S_{30}), are depicted in the Figure 4.8.

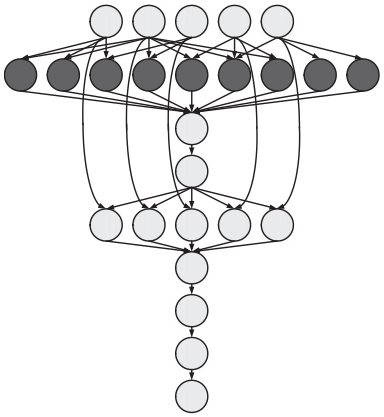
To simulate the makespan slowdown of the multiprogrammed cluster nodes (DP=2), we have generated a slowdown matrix similar to the one of Table 2.5, but with as many rows and columns as possible combinations of two tasks in Epigenomics, Montage, and Sipht workflows. Each cell in the matrix has been given random numbers between 0 and 1 (0% and 100%). This way, we can show the benefits of multiprogramming in Workflowsim clusters.

Table 4.12 Multiworkflows processed to validate HBRM with the SA and SAFP algorithms on large clusters.

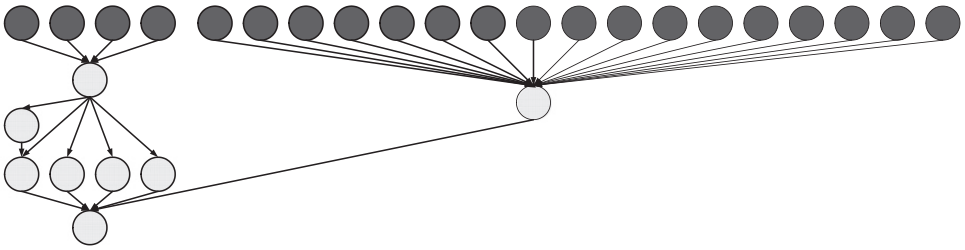
Multiworkflows Processed	Workflow replicated	# Instances or replicas	Total amount of tasks of the multiworkflow
MWF_1	Epigenomics 24 tasks	200	4800
MWF_2	Epigenomics 100 tasks	20	2000
MWF_3	Epigenomics 997 tasks	5	4985
MWF_4	Montage 25 tasks	200	5000
MWF_5	Montage 100 tasks	50	5000
MWF_6	Montage 1000 tasks	5	5000
MWF_7	Sipht 30 tasks	100	3000
MWF_8	Sipht 100 tasks	30	3000
MWF_9	Sipht 1000 tasks	4	4000



(a) Epigenomics Workflow with 24 tasks (E_{24}).



(b) Montage Workflow with 25 tasks (M_{25}).



(c) Sipht Workflow with 30 tasks (S_{30}).

Figure 4.8 Structures of the Epigenomics, Montage and Sipht workflows.

4.5.2 Resources

The experiments are conducted large clusters with amounts of nodes ranging from 64 to 1024. As explained in Section 3.2, to generate a cluster-like environment with Workflowsim we used one datacenter, and each physical machine or host has been equipped with a single VM, which represents a cluster node. Each VM, contains multiple PUs, and can host the execution of up to two applications. This allows us to carry out shared-node execution, necessary to apply the SA

and SAFP algorithms, and generate the environment to apply the slowdown to the overlap time depicted in Figure 3.3.

The main specifications of the cluster generated for the experiments are included in Table 4.13

Table 4.13 Specifications of the large clusters generated for the HBRM validation with SA and SAFP algorithms.

	Simulator Specifications
Nodes p/Cluster	64, 128, 256, 512, 1024
Processors p/Node	32 PUs (2CPUs)
Processors Freq.	6000 MIPS
RAM p/Node	96 GB
Disk capacity p/Node	1 TB
Net latency	0.2 ms
Internal latency	0.05 ms

The memory hierarchy is structured differently depending on the scheduling algorithms used. SA, Max-Min and Max-Min dispose of the default Workflowsim memory hierarchy. Each node has a RAM and a local hard disk. All cluster nodes share a NFS unit of theoretically unlimited size.

To run the SAFP algorithm, the original memory hierarchy of Workflowsim has been modified. Each node contains a RAM over which a local RAM disk has been mounted, and a local SSD. The NFS unit remains unaltered.

Table 4.14 Main specifications of the storage systems employed for the simulation.

Storage	Mounted Over	Latency	Average Seek Time	Maximum Transfer Rate	Size
NFS	HDD 7200 RPM	8 ms	25 ms	100 MBs	unlimited
Local Disk	SSD	0.06 ms	0.1 ms	500 MBs	200 GB/VM
Local Ramdisk	DDR3 SDRAM-1333	0.003 ms	0.001 ms	1.8 GBs	6 GB/VM

4.5.3 Overview of the experiments and analysis of the results

To validate the HBRM on large clusters with the SA and SAFP algorithm we process a workload formed by 9 multiworkflows, generated with multiple instances of Epigenomics (24, 100, and 997 tasks), Montage (25, 100, and 1000 tasks), and Sipht (30, 100, and 1000 tasks). In each large cluster, the multiworkflows are processed with the following algorithms: HBRM's SA and SAFP, Min-Min and Max-Min. At the end, the makespans of each scheduling policy are compared and discussed. The summarized layout of the experiments is represented in Figure 4.9.

As explained with more detail in Section 4.3.2, all the validation experiments in Workflowsim have been conducted with DP=2. To allocate two different tasks in the same node, we have

followed the approach shown rightmost part of Figure 3.2. Each PU has been equipped with 2 CPUs, and in each CPU has been placed one task.

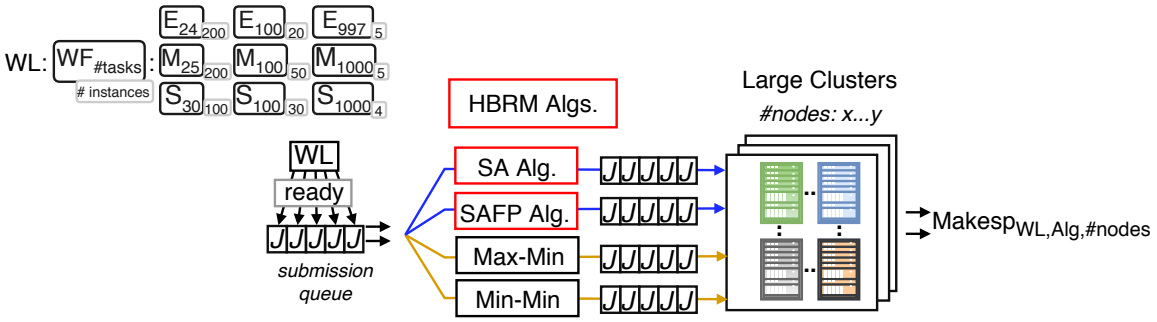


Figure 4.9 Layout of the experiments conducted on large clusters to validate the HBRM with the SA and SAFP algorithms. A workload of 9 different multiworkflows, generated with multiple instances (within gray rectangles) of Epigenomics (E), Montage (M), and Sipth (S) workflows is processed with four scheduling algorithms: SA, SAFP, Max-Min, and Min-Min, on clusters with different number of nodes.

Each of the 9 multiworkflows processed is defined in a different XML file, and then handed to Workflowsim for processing. Thus, one file for Epigenomics with 24 tasks, (E_{24}), another for Epigenomics with 46 tasks (E_{46}), and so on.

The XML files containing the 9 multiworkflows are included in the Workflowsim's default package, and come along with makespan estimations. Therefore, the prediction block of the HBRM is not applied for these particular experiments.

Workflows may be dynamically submitted to clusters. As explained in Section 4.1, the dynamic processing is turned into static by defining batches of workflows. In this case, involving dozens of different workflows and node resources, numerous batches have been generated by including multiple instances of the same workflows.

Before starting execution, all cluster nodes are idle, and all multiworkflows' required input files are by Workflowsim loaded onto the shared NFS storage unit. After going through the prediction phase, the admitted multiworkflows are scheduled by different algorithms: HBRM's SA and SAFP, and state-of-the-art Max-Min and Min-Min. As explained in 4.1, the DP of the nodes is 2, and their status can be idle (0 tasks running and all PUs idle), loaded (1 task running, some PUs busy and some idle), and busy (2 tasks running and all PUs busy).

- The Min-Min algorithm calculates the completion overall time of all admitted tasks, prioritizes the shortest one, and assigns it to the node in which the task is expected to yield minimum completion time.

- The Max-Min algorithm proceeds similarly to Min-Min, albeit it sorts the admitted tasks from maximum to minimum overall time and prioritizes the longest one. For each validation test, obtained average workflow makespans are reviewed and compared.

- At the beginning of the execution, the SA algorithm sorts the applications from longest to shortest, and places them in idle nodes. Once all nodes are loaded (with some PUs idle and some other busy), the SA algorithm calculates the slowdown information of all combinations of the list of admitted tasks. At the beginning of the processing, schedules the longest applications in the idle nodes, until all are loaded. Then calculates, the amount of slowdowns between each combination of queued and running tasks, and allocates the queued task to the loaded node where the minimum slowdown is yielded.

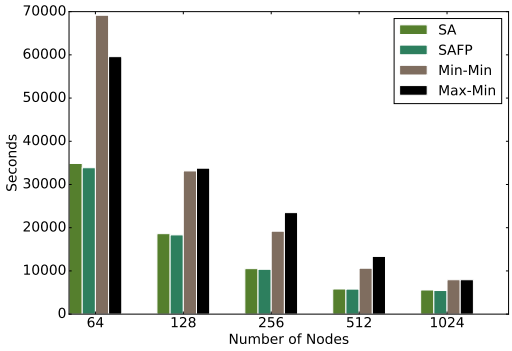
- The SAFP algorithm proceeds similarly to the SA algorithm, but considers the different memory hierarchy levels when placing the required files of the multiworkflow. It considers whether files are shared by more than one task. It also considers and three different file sizes: small (<1MB), medium (between 1MB and 1GB), and big (>1GB). Small files and shared medium files are placed in the local RAM disk of the node hosting the execution, medium files and big shared files are placed in the local disk of the nodes, big non-shared files are kept in the NFS unit.

One of the advantages of the SA and SAFP algorithms is that their algorithms consider the degree of multiprogramming of the nodes (DP>1). They generate the most compatible combinations of applications, and schedule applications in loaded (partially used) nodes so that the resource competition and thus slowdown are minimized. Conversely, Min-Min and Max-Min (also adapted to be tested with DP=2) schedule combinations of applications for shared same-node execution by dismissing the amount of makespan slowdown yielded. Furthermore, the advantage of SA algorithm over SAFP algorithm is that it considers the memory hierarchy when placing the different files required by the multiworkflows.

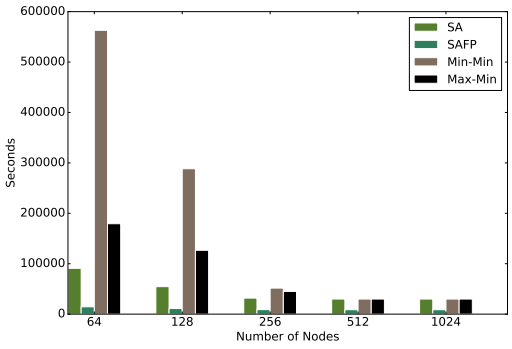
After the processing of the 9 multiworkflows in the different large clusters is completed, makespan results are obtained. First, we will review the makespan results of the Epigenomics multiworkflows (MWF_1, MWF_2, and MWF_3), second, the makespan results of the Montage multiworkflows (MWF_4, MWF_5, and MWF_6), and third, the makespan results of the Epigenomics multiworkflows (MWF_7, MWF_8, and MWF_9).

In Figure 4.10 (a), Figure 4.10 (b), and Figure 4.11 we can see the makespans obtained after processing on large clusters the Epigenomics multiworkflows with different tasks and instances: MWF_1: ($E_{24tasks}$ x 200 instances), MWF_2: ($E_{100tasks}$ x 20 instances), and MWF_3: ($E_{997tasks}$ x 5 instances), respectively.

As we can see in Figure 4.10 (a), Figure 4.10 (b), and Figure 4.11, the multiworkflow makespans obtained with the HBRM algorithms (SA and SAFP) outperform by far the ones obtained with state of the art Min-Min and Max-Min policies in the wide range of clusters the testing has taken place. In Table 4.15, we summarize the Epigenomics multiworkflows (MWF_1, MWF_2 and MWF_3) makespan improvements obtained with SA and SAFP algorithm versus Min-Min and Max-Min, as well as the improvement of SA versus SAFP.



(a) Makespans obtained after processing MWF_1: 200 instances of Epigenomics workflow with 24 tasks (E_{24}).



(b) Makespans obtained after processing MWF_2: 20 instances of Epigenomics workflow with 100 tasks (E_{100}).

Figure 4.10 Makespans obtained after processing two different Epigenomics multiworkflows with different scheduling policies and different clusters.

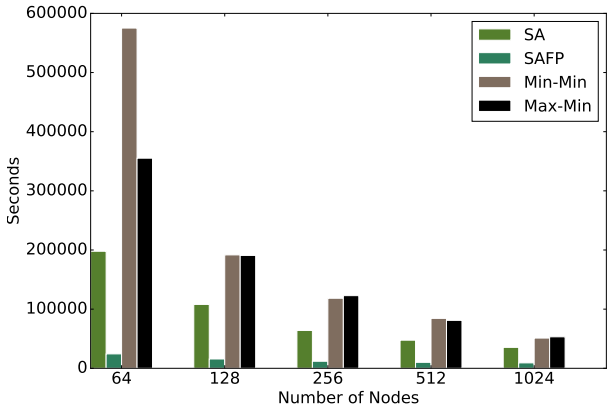


Figure 4.11 Makespans obtained after processing MWF_3, a workload composed by 5 instances of Epigenomics workflow with 997 tasks (E_{997}) on clusters of up to 1024 nodes with different scheduling policies.

Table 4.15 shows how the Epigenomics multiworkflow makespans obtained with SA and SAFP algorithms improves the makespan of Min-Min and Max-Min. SA outperforms Min-Min and Max-Min by 40% on average, and SAFP outperforms Min-Min and Max-Min by 63%, on average. Finally, SAFP improves the average makespan of SA by 53%. The conclusions extracted from the processing of MWF_1, MWF_2 and MWF_3 are:

- SA algorithm shows 40% MWF makespan improvement versus Min-Min and Max-Min, which is steady across the three different multiworkflows.
- SAFP notoriously improves the makespans of SA, for the MWF_2 (76%) and MWF_3 (82%). However, for MWF_1, the performance of SA and SAFP is almost identical, varying 20%. This is because barely any input file of the Epigenomics workflow of 24 tasks can benefit from the RAM disk. The requirements for files to be placed in the RAM disk is: to be small (<1MB),

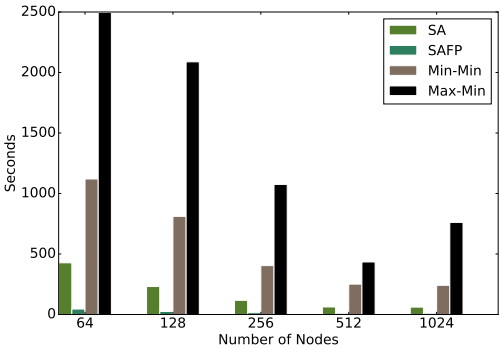
and Epigenomics 24 only includes small file, or to be shared among >1 task, and Epigenomics only has 4 files out of 34 that are shared.

- As the number of files that comply with the requirements to be placed in RAM disk increases, the performance of SAFP increasingly outperforms SA, reaching up to 82% improvement. Similarly, the improvement of SAFP over Min-Min and Max-Min reaches up to 89%.

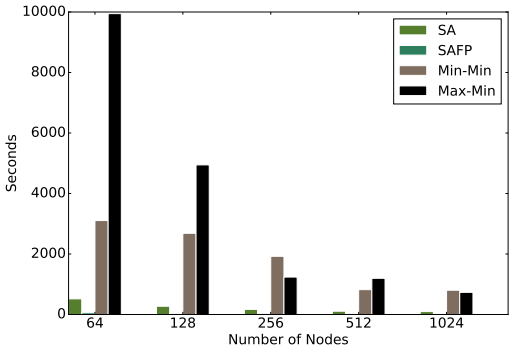
Table 4.15 Summary of the results obtained when processing Epigenomics multiworkflows.

Epigenomics MWFs	Improvement of SA vs. Av. (Max-Min, Min-Min)	Improvement of SAFP vs. SA	Improvement of SAFP vs. Av. (Max-Min, Min-Min)
MWF_1	44%	2 %	45%
MWF_2	34%	76 %	63 %
MWF_3	44%	82 %	89%
Av.	*40%*	*53%*	*63%*

In Figure 4.12 (a), Figure 4.12 (b), and Figure 4.13, we can see the results obtained after processing on large clusters the Montage workflow with different tasks and instances: MWF_4: ($M_{25tasks}$ x 200 instances), MWF_5: ($M_{100tasks}$ x 50 instances), and MWF_6: ($M_{1000tasks}$ x 5 instances), respectively.



(a) Makespans obtained after processing MWF_4: 200 instances of Montage workflow with 25 tasks (M_{25}).



(b) Makespans obtained after processing MWF_5: 20 instances of Montage workflow with 100 tasks (M_{100}).

Figure 4.12 Makespans obtained after processing two different Montage multiworkflows with different scheduling policies and different clusters.

As we can see in Figure 4.12 (a), Figure 4.12 (b), and Figure 4.13, the multiworkflow makespans obtained with the HBRM algorithms (SA and SAFP) outperform by far the ones obtained with state of the art Min-Min and Max-Min policies in the wide range of clusters the testing has taken place. In the Table 4.16, we summarize the Montage multiworkflows (MWF_4, MWF_5 and MWF_6) makespan improvements obtained with SA and SAFP algorithm versus Min-Min and Max-Min, as well as the improvement of SA versus SAFP.

Table 4.16 shows how the Montage multiworkflow makespans obtained with SA and SAFP algorithms improve the makespan of Min-Min and Max-Min. SA outperforms Min-Min and Max-Min by 82% on average, and SAFP outperforms Min-Min and Max-Min by 96%, on average.

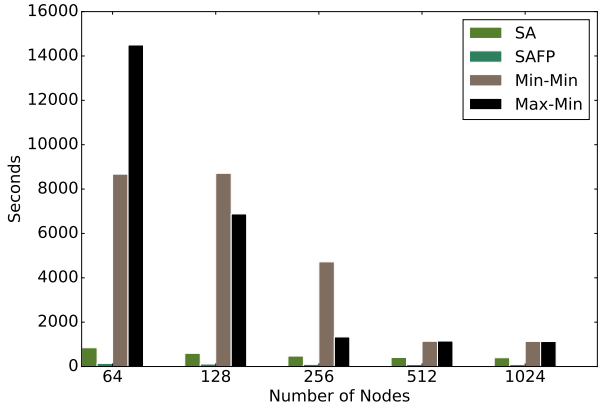


Figure 4.13 Makespans obtained after processing MWF_6, a workload composed by 5 instances of Montage workflow with 1000 tasks (M_{1000}) on clusters of up to 1024 nodes with different scheduling policies.

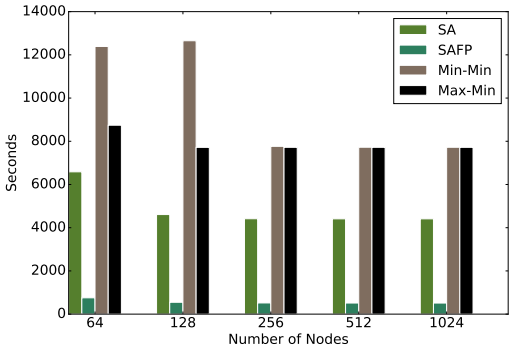
Finally, SAFP improves the average makespan of SA by 84%. The conclusions extracted from the processing of MWF_4, MWF_5 and MWF_6 are:

- SA and SAFP algorithms show a 82% and 96% MWF makespan improvements versus Min-Min and Max-Min. The improvements over the HBRM algorithms are due to the consideration of the degree of the multiprogramming of the nodes, which minimizes the resource competition and makespan slowdown. Conversely, Min-Min and Max-Min, schedule combinations of tasks for shared-node execution regardless of the slowdown, which may then soar, abruptly lengthening the multiworkflow makespan.
- In turn, SAFP notoriously improves the makespans of SA, in an even fashion: 87% for MWF_4, 86% for MWF_5, and 80% for MWF_6, averaging 84%
- The multiworkflow makespan of SAFP versus Min-Min and Max-Min shows a steady increase for the three Montage multiworkflows, averaging 96%.

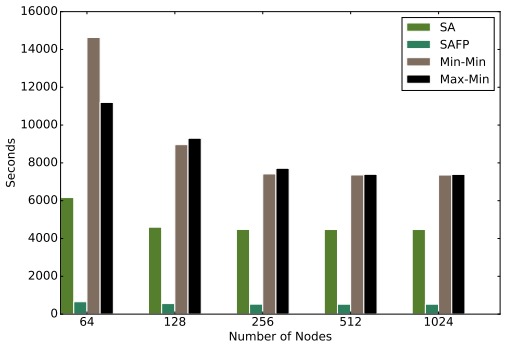
Table 4.16 Summary of the results obtained when processing Montage multiworkflows.

Montage MWFs	**Improvement of SA vs. Av. (Max-Min, Min-Min)**	**Improvement of SAFP vs. SA**	**Improvement of SAFP vs. Av. (Max-Min, Min-Min)**
MWF_4	77%	87 %	97%
MWF_5	90%	86 %	97 %
MWF_6	78%	80 %	95%
Av.	*82%*	*84%*	*96%*

In Figure 4.14 (a), Figure 4.14 (b), and Figure 4.15, we can see the results obtained after processing on large clusters the Sipht (S) workflow with different tasks and instances: MWF_7: ($S_{30tasks}$ x 100 instances), MWF_8: ($S_{100tasks}$ x 30 instances, and MWF_9: ($S_{1000tasks}$ x 4 instances), respectively.



(a) Makespans obtained after processing MWF_7: 100 instances of Sipht workflow with 30 tasks (S_{30}).



(b) Makespans obtained after processing MWF_8, 30 instances of Sipht workflow with 100 tasks (S_{100}).

Figure 4.14 Makespans obtained after processing two different Sipht multiworkflows with different scheduling policies and different clusters.

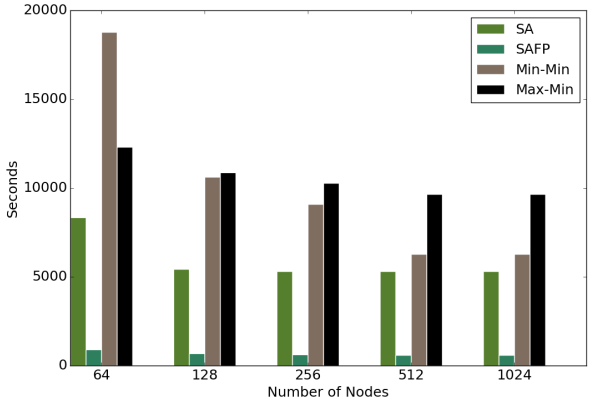


Figure 4.15 Makespans obtained after processing MWF_9, a workload composed by 4 instances of Sipht workflow with 1000 tasks (S_{1000}) on clusters of up to 1024 nodes with different scheduling policies.

As we can see in Figure 4.14 (a), Figure 4.14 (b), and Figure 4.15, the multiworkflow makespans obtained with the HBRM algorithms (SA and SAFP) outperform by far the ones obtained with state of the art Min-Min and Max-Min policies in the wide range of clusters the testing has taken place.

In the Table 4.17, we summarize the Sipht multiworkflows (MWF_7, MWF_8 and MWF_9) makespan improvements obtained with SA and SAFP algorithm versus Min-Min and Max-Min, as well as the improvement of SA versus SAFP. Table 4.17 shows how the Montage multiworkflow makespans obtained with SA and SAFP algorithms improve the makespans of Min-Min and Max-Min. SA outperforms Min-Min and Max-Min by 43 % on average, and SAFP outperforms Min-Min and Max-Min by 93%, on average. Finally, SAFP improves the average makespan of SA by 88 %. The conclusions extracted from the processing of MWF_7, MWF_8 and MWF_9 are:

- SA and SAFP algorithms generate 43% and 93% MWF makespan improvements versus

Min-Min and Max-Min, respectively. The improvements over the HBRM algorithms are due to the consideration of the degree of the multiprogramming of the nodes, which minimizes the resource competition and makespan slowdown. Conversely, Min-Min and Max-Min, schedule combinations of tasks for shared-node execution regardless of the slowdown, which may then soar, abruptly lengthening the multiworkflow makespan.

- In turn, the SAFP notoriously improve the makespans of SA, in a constant manner: 88% for MWF_7, MWF_8, MWF_9.
- The multiworkflow makespan of SAFP versus Min-Min and Max-Min shows a steady increase for the three Montage multiworkflows, averaging 96%.

Table 4.17 Summary of the validation results obtained when processing Sipht multiworkflows.

Sipht MWFs	Improvement of SA vs. Av. (Max-Min, Min-Min)	Improvement of SAFP vs. SA	Improvement of SAFP vs. Av. (Max-Min, Min-Min)
MWF_7	43%	88 %	93%
MWF_8	44%	88 %	93 %
MWF_9	42%	88 %	93%
Av.	*43%*	*88%*	*93%*

In the validation of the HBRM on large clusters done with 9 different multiworkflows made with different instances of Epigenomics, Montage and Sipht, we have proved how the SA and SAFP algorithms improve the multiworkflow by large makespans of Min-Min and Max-Min algorithms. The average improvements for all multiworkflows are shown in Table 4.18. On average, HBRM's SA algorithm improves the 9 multiworkflow makespans of Min-Min and Max-Min by 68 %. The 68% multiworkflow makespan improvement is attained considering, in each scheduling step, the makespan slowdown of tasks in multiprogrammed nodes. HBRM's SAFP algorithm improves the 9 multiworkflow makespans of Min-Min and Max-Min by 84 %, and the makespans of SA by 88%. The improvements that SAFP, which also accounts for the multiprogramming slowdown, achieves over SA, are due to the consideration of the different levels of the memory hierarchy (local RAM disk, local disk, or NFS) when deciding where the different input files of the workflow tasks are placed. The decisions are taken upon the size of the file, and whether the file is shared by multiple tasks or not.

Table 4.18 Summary of the validation results with all 9 multiworkflows processed.

Epigen., Montage, Sipht	Improvement of SA vs. Av. (Max-Min, Min-Min)	Improvement of SAFP vs. SA	Improvement of SAFP vs. Av. (Max-Min, Min-Min)
Av.Epigenomics MWFs	40%	63 %	53%
Av.Montage MWFs	82%	96 %	54 %
Av.Sipht MWFs	83%	93 %	58%
Av. 9 WMFs	*43%*	*88%*	*93%*

Chapter 5

Conclusions and Future Research lines

In this work we have introduced a practical solution to manage the resources of shared clusters hosting bioinformatics workflows applications. Throughout this work we identified the most relevant factors or reasons why the current approaches taken by current RMS to allocate tasks to resources are not adequate in environments mostly hosting bioinformatics tasks.

This has helped us divide the main problem into different parts, which represent the conflicting points we have addressed and tackled. Thorough study of these points has helped us establish the major lines of work required to deploy the different blocks of the HBRM.

In this section we expose the conclusions extracted once completed all the steps exposed in this work. We start by mentioning the publications done over these years, which allowed us entrench the advancements accomplished, as well as enhancing the HBRM.

Next, we expose the conclusions extracted once completed the framework of the proposed HBRM. Eventually, we mention the future lines of work considered to further enhance the proposed resource manager.

5.1 Publications

The articles published throughout this thesis are introduced in this section.

In the first publication, titled: A Resource Manager for Maximizing the Performance of Bioinformatics Workflows in Shared Clusters, we defined the foundations of the HBRM and its main blocks: Application Characterization, the Multivariate Regression Prediction Model, and the Multicriteria Scheduler. We validated the HBRM on clusters with the SA algorithm, and improved the average makespans, resource usages and efficiencies of bioinformatics workflow applications obtained with SLURM's FCFS. This paper was selected as one of the Best Papers of the International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP-2017). Due to this qualification, we were invited to present an extended version of our work in the International Journal of Parallel Programming (2016 JCR impact factor: 1.156, Quartile Q3).

Authors: ***Ferran Badosa**, César Acevedo, Antonio Espinosa, Gonzalo Vera, Ana Ripoll.*
Title: **A Resource Manager for Maximizing the Performance of Bioinformatics Workflows in Shared Clusters.**
17^{th} International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP-2017), 5^{th} International Workshop on Parallelism in Bioinformatics (PBio).
Helsinki, Finland, 2017.

In the second publication, titled: Bio-backfill: a scheduling policy enhancing the performance of bioinformatics workflows in shared clusters, we designed a new scheduling policy for the HBRM, the Biobackfill scheduling policy. In this publication, we validated how the HBRM's Biobackfill policy improved the average workflow makespan obtained with Firstfit and Bestfit policies. In this publication, we first experimented with the Workflowsim simulator.

Authors: ***Ferran Badosa**, Antonio Espinosa, Gonzalo Vera, Ana Ripoll.*
Title: **Bio-backfill: a scheduling policy enhancing the performance of bioinformatics workflows in shared clusters.**
International Conference on Complexity, Future Information Systems and Risk (COMPLEXIS).
Madeira, Portugal, 2018.

The third publication, entitled A history-based Resource Manager for Genome Analysis Workflows Applications on clusters with heterogeneous nodes, has been published in the International Journal on Parallel Programming (IJPP), in which we were offered to publish after being chosen among the Best Papers of the ICA3PP-2017 conference. In this journal article, we covered in depth the extensions made in Workflowsim to adapt it to the necessary requirements to implement the HBRM's SA algorithm. We validated the modifications by testing that no flaws had been introduced. Also, we validated the HBRM's SA algorithm on large clusters by proving it improved the average workflow makespan obtained with Min-Min and Max-Min.

Authors: ***Ferran Badosa**, Antonio Espinosa, César Acevedo, Gonzalo Vera, Ana Ripoll.*
Title: **A history-based Resource Manager for Genome Analysis Workflows Applications on clusters with heterogeneous nodes.**
D.O.I.: 10.1007/s10766-018-0600-z
International Journal on Parallel Programming (IJPP).
2018.

5.2 Conclusions

The main conclusions extracted after completing this thesis are listed below:

- Current RMS on clusters consider the characteristics and resource consumptions of the majority of applications running on clusters to minimize the consequences of potential performance issues and improve workflows' performance. Among these one may find distributed-memory applications programmed with MPI which are executed in multiple nodes at a time, and whose performance is commonly bounded by their and have large communication requirements. Also, the performance of most applications barely varies upon parameter values, for it is not significantly difficult for managers to determine the resources needed. These approaches fail to suit scenarios mostly hosting bioinformatics applications. They are commonly programmed in shared-memory paradigms, and pose different performance issues, which prompt them to show low scalability. On the other hand, they may contain different algorithms within, triggering different resource usages upon the parameter values specified and datasets chosen. Due to that, current RMS approaches don't adjust to bioinformatics' applications profiles, compromising applications' performance and squandering cluster resources.

- History-based approaches approaches to manage resources are suitable for estimating the resources needed by applications in future runs. However, as doing so, it must be considered the variable behavior of bioinformatics applications, which prompt them to have significantly different resource requirements, consumptions and makespans upon parameters and data. Developing a history-based approach aware of the characteristics of applications (parameters and datasets), has proven beneficial in order to establish a prediction model that harnesses performance information from past runs upon which can be allocated resources in future similar runs.

- Accounting for the resource usage of applications when deciding which combinations of applications are more compatible for sharing the multiprogrammed nodes can significantly reduce the amount of makespan slowdown yielded, which in turn decreases the average makespans of the workflows.

- The aforementioned informations, the performance predictions of applications given parameters and data, and the slowdown of applications when multiprogramming nodes, can be harnessed for new scheduling algorithms to be coded. A scheduling algorithm accounting for the resource usage profile of applications in both exclusive and shared mode improves overall performance of applications and makes the most of resources. Resources otherwise reserved and low utilized such as PUs from a node, are available for other applications, who see their waiting times reduced and can make full utilization of these PUs.

- The proposed resource manager, HBRM, composed of the Application characterization, Multivariate Regression Prediction and a Scheduler Multicriteria, can improve the makespans of the bioinformatics workflows submitted to shared clusters.

  The HBRM implemented on clusters with the SA algorithm can improve the average workflow makespan by up to 34%, the average resource utilization by 86%, and the average resource efficiency by 96% obtained with SLURM's FCFS.

  The HBRM implemented on clusters with the Biobackfill algorithm can improve the average workflow makespan by 8.55%, compared to Firstfit and Bestfit.

  The HBRM with the SA algorithm implemented on large clusters, and tested under heavily-loaded conditions (different Epigenomics, Montage and Sipht multiworkflows of thousands of tasks), can improve on average, the multiworkflow makespan by 43%, compared with Min-Min and Max-Min algorithms.

- The size of files required bioinformatics workflow applications is significantly greater than those of most applications running on clusters. Generating a scheduling policy that not only considers performance predictions in exclusive mode and slowdown from node multiprogramming, but also considers file characteristics such as size and whether they're shared, can help reduce the makespans of applications. In turn, memory utilization by properly placing the different required workflow files in the memory hierarchy levels of clusters, such as local RAM disks or local disks.

  The HBRM with the SAFP implemented on large clusters, and tested under heavily-loaded conditions (different Epigenomics, Montage and Sipht multiworkflows of thousands of tasks), can improve on average, the multiworkflow makespan by 93%, compared with Min-Min and Max-Min algorithms.

  The HBRM's SAFP algorithm implemented on large clusters, and tested under heavily-loaded conditions (different Epigenomics, Montage and Sipht multiworkflows of thousands of tasks), can improve on average, the multiworkflow makespan by 88%, obtained with the HBRM's SA algorithm.

- The backbone of the HBRM is the prediction model, the founding stone upon which all steps. In turn the prediction model analyzes information obtained from past runs and stored in the database. The more information gathered, the better tools for the prediction model to generate estimations. Implementing a feedback model is paramount for the manager. It allows for performance of information for each run to be stored in the historical database, and possess a ever-growing volume of data continually widening the knowledge of the HBRM. Also, with the feedback, once a run is completed, the predictor can compare prediction and newly-obtained performance results, and correct the prediction error. With that feature the predictor is dynamically updated, and yields improved accuracy.

- Encompassing the different lines of work (characterization, prediction, scheduling...) in a resource managing approach such as the HBRM stands as a powerful alternative against the traditional approaches of current RMS. The increasing relevance of bioinformatics in the current times and the need to efficiently process enormous amounts of information emphasizes the relevance of the work deployed, raising awareness of the substantial benefits the HBRM can provide in shared platforms for biological data analysis such as clusters.

5.3 Future Research lines

The importance of the resource allocation problem and the scarce solutions so far proposed in the literature for the particular case of bioinformatics applications suggest there's a lengthy path ahead of us to be explored. The future lines considered after reaching a point where the HBRM has proven useful are listed below

- Although the characterization approach includes a representative set of applications and is applicable to many others, it would be interesting to increase the amount of applications to test the model.

- This work considers the DP of the nodes when sharing clusters. Optimization of the DP considering all the factors involved: applications, resources, datasets and node resource specification, can trigger potential benefits for overall execution

- Integrate the HBRM into a SLURM plug-in by using the SLURM spank interface for the development of plug-ins. We have proved the HBRM to be compatible for operation alongside SLURM, one of the top default RMS on clusters. Encapsulating the HBRM into a SLURM plug-in would favor the deployment of the work done on clusters.

- Optimization of Workflowsim for the SA and SAFP algorithms. Workflowsim blocks have been modified and adapted to the requirements of the mentioned algorithms. Adaptation allowed for the simulator to fulfill the needs of both algorithms, such as presenting nodes as half-used, with some PUs busy and some other idle, complete monitoring of the nodes' load (enabling to find out whether and where applications scheduled in past scheduling iterations are running), and implementation of slowdown. However, there is still margin for including in it further enhancements that would allow for new scheduling approaches to be implemented.

Bibliography

[1] SLURM. URL www.slurm.schedmd.com/job{_}array.html.

[2] National Center for Biotechnology Information, 2017. URL https://www.ncbi.nlm.nih.gov/genbank/statistics/.

[3] DNA Sequencing Costs: Data, 2017. URL https://www.genome.gov/27541954/dna-sequencing-costs-data/.

[4] César Acevedo, Porfidio Hernández, Antonio Espinosa, and Víctor Méndez. A Critical Path File Location (CPFL) algorithm for data-aware multiworkflow scheduling on HPC clusters. *Future Generation Computer Systems*, 74:51–62, 2017.

[5] S F Altschul and B W Erickson. Significance of nucleotide sequence alignments: a method for random sequence permutation that preserves dinucleotide and codon usage. *Molecular biology and evolution*, 2(6):526–538, 1985.

[6] Rashmi Bajaj and Dharma P. Agrawal. Improving Scheduling of Tasks in a Heterogeneous Environment. *IEEE Transactions on Parallel and Distributed Systems*, 15(2):107–118, 2004.

[7] Giacomo Baruzzo, Katharina E. Hayer, Eun Ji Kim, Barbara DI Camillo, Garret A. Fitzgerald, and Gregory R. Grant. Simulation-based comprehensive benchmarking of RNA-seq aligners. *Nature Methods*, 14(2):135–139, 2017.

[8] Shishir Bharathi, Ann Chervenak, Ewa Deelman, Gaurang Mehta, Mei Hui Su, and Karan Vahi. Characterization of scientific workflows. *2008 3rd Workshop on Workflows in Support of Large-Scale Science, WORKS 2008*, 2008.

[9] Ivan Borozan, Stuart N. Watt, and Vincent Ferretti. Evaluation of alignment algorithms for discovery and identification of pathogens using RNA-Seq. *PloS one*, 8(10), 2013.

[10] Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82–117, 2013.

[11] Anca I D Bucur and Dick H J Epema. The Influence of Communication on the Performance of Co-allocation. In *7th International Workshop on Job Scheduling Strategies for Parallel Processing*, pages 66–86, 2001.

[12] R Buyya and M Murshed. GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. *Concurrency and Computation-Practice & Experience*, 14(13-15):1175–1220, 2002.

[13] Henri Casanova. Simgrid: A toolkit for the simulation of application scheduling. *Proceedings - 1st IEEE/ACM International Symposium on Cluster Computing and the Grid, CCGrid 2001*, pages 430–437, 2001.

[14] Weiwei Chen and Ewa Deelman. WorkflowSim: A toolkit for simulating scientific workflows in distributed environments. *2012 IEEE 8th International Conference on E-Science, e-Science 2012*, 2012.

[15] Peter J.A. Cock, Björn A. Grüning, Konrad Paszkiewicz, and Leighton Pritchard. Galaxy tools and workflows for sequence analysis with applications in molecular plant pathology. *PeerJ*, 1:e167, 2013.

[16] Peter A. Dinda. Online prediction of the running time of tasks. *Cluster Computing*, 5(3): 225–236, 2002.

[17] Fangpeng Dong and Selim G Akl. Scheduling Algorithms for Grid Computing : State of the Art and Open Problems. *Components*, 202(4):1–55, 2006.

[18] A.B. Downey. Predicting queue times on space-sharing parallel computers. *Proceedings 11th International Parallel Processing Symposium*, (July):209–218, 1997.

[19] Catalin L. Dumitrescu and Ian Foster. GangSim; A simulator for grid scheduling studies. *2005 IEEE International Symposium on Cluster Computing and the Grid, CCGrid 2005*, 2 (June):1151–1158, 2005.

[20] Joseph Felsenstein. Confidence Limits on Phylogenies: An Approach Using the Bootstrap. *Evolution*, 39(4):783, 1985.

[21] Silvia M. Figueira and Francine Berman. A slowdown model for applications executing on time-shared clusters of workstations. *IEEE Transactions on Parallel and Distributed Systems*, 12(6):653–670, 2001.

[22] César Gómez-Martín, Miguel A. Vega-Rodríguez, and José Luis González-Sánchez. Fattened backfilling: An improved strategy for job scheduling in parallel systems. *Journal of Parallel and Distributed Computing*, 97:69–77, 2016.

[23] Charlotte Herzeel, Thomas J Ashby, Charlotte Herzeel, and Pascal Costanza. Performance Analysis of BWA Alignment. 2013.

[24] Charlotte Herzeel, Thomas J. Ashby, Pascal Costanza, and Wolfgang De Meuter. Resolving load balancing issues in BWA on NUMA multicore architectures. *Lecture Notes in Computer Science*, 8385 LNCS(2):227–236, 2014.

[25] Adan Hirales-Carbajal, Andrei Tchernykh, Thomas Röblitz, and Ramin Yahyapour. A Grid Simulation Framework to Study Advance Scheduling Strategies for Complex Workflow Applications. In *Proceedings of the 24th IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW'10)*, pages 1–8, 2010.

[26] Jing-Jang Hwang, Yuan-Chieh Chow, Frank D. Anger, and Chung-Yee Lee. Scheduling Precedence Graphs in Systems with Interprocessor Communication Times. *SIAM Journal on Computing*, 18(2):244–257, 1989.

[27] Alexandru Iosup, Hui Li, Mathieu Jan, Shanny Anoep, Catalin Dumitrescu, Lex Wolters, and Dick H.J. Epema. The Grid Workloads Archive. *Future Generation Computer Systems*, 24(7):672–686, 2008.

[28] Nagarajan Kathiresan, Ramzi Temanni, Hakeem Almabrazi, Najeeb Syed, Puthen V. Jithesh, and Rashid Al-Ali. Accelerating next generation sequencing data analysis with system level optimizations. *Scientific Reports*, 7(1):1–11, 2017.

[29] Ravneet Kaur and Ramneek Kaur. Multiprocessor Scheduling Using Task Duplication Based Scheduling Algorithms : A Review Paper. *International Journal of Application or Innovation in Engineering & Management*, 2(4):311–317, 2013.

[30] Istvan Ladunga. Finding similar nucleotide sequences using network BLAST searches. *Current Protocols in Bioinformatics*, 2017:3.3.1–3.3.25, 2017.

[31] Ben Langmead. Aligning short sequencing reads with Bowtie. *Current Protocols in Bioinformatics*, (SUPP.32):1–14, 2010.

[32] Jeremy Leipzig. A review of bioinformatic pipeline frameworks. *Briefings in Bioinformatics*, 18(3):530–536, 2017.

[33] Heng Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. 00(00):1–3, 2013. URL `http://arxiv.org/abs/1303.3997`.

[34] Kevin Liu, C. Randal Linder, and Tandy Warnow. RAxML and FastTree: Comparing two methods for large-scale maximum likelihood phylogeny estimation. *PLoS ONE*, 6(11), 2011.

[35] Muthucumaru Maheswaran, Shoukat Ali, Howard Jay Siegel, Debra Hensgen, and Richard F Freund. Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous ... *Journal of Parallel and Distributed Computing*, 131(June 1999):107–131, 1999.

[36] Rahul Malhotra and Prince Jain. Study and Comparison of Various Cloud Simulators Available in the Cloud Computing. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(9):347–350, 2013.

[37] Munir Merdan, Thomas Moser, Dindin Wahyudin, Stefan Biffl, and Pavel Vrba. Simulation of workflow scheduling strategies using the MAST test management system. *2008 10th International Conference on Control, Automation, Robotics and Vision, ICARCV 2008*, (December):1172–1177, 2008.

[38] Ajith Singh N and M Hemalatha. High performance computing network for cloud environment using simulators. *arXiv preprint arXiv:1203.1728*, page 5, 2012.

[39] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similiarities in the amino acid sequence of two proteins, 1970.

[40] Chad Nelson, Kevin Townsend, Bhavani Satyanarayana Rao, Phillip Jones, and Joseph Zambreno. Shepard : A Fast Exact Match Short Read Aligner. (July 2012), 2015.

[41] Corey B. Olson, Maria Kim, Cooper Clauson, Boris Kogon, Carl Ebeling, Scott Hauck, and Walter L. Ruzzo. Hardware Acceleration of Short Read Mapping. *Proceedings of the 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines, FCCM 2012*, pages 161–168, 2012.

[42] Christian Otto, Peter F. Stadler, and Steve Hoffmann. Lacking alignments? The next-generation sequencing mapper segemehl revisited. *Bioinformatics*, 30(13):1837–1843, 2014.

[43] Harshadkumar B. Prajapati and Vipul A. Shah. Scheduling in Grid Computing Environment. *2014 Fourth International Conference on Advanced Computing & Communication Technologies*, pages 315–324, 2014.

[44] Prakash Murali and Sathish Vadhiyar. HPCTOOLKIT: Tools for performance analysis of optimized parallel programs. *Concurrency Computation Practice and Experience*, 22(6): 685–701, 2010.

[45] Radu Prodan. Specification and runtime workflow support in the ASKALON Grid environment. *Scientific Programming*, 15(4):193–211, 2007.

[46] Benjamin Quetier and Franck Cappello. A survey of Grid research tools: simulators, emulators and real life platforms. In *IMACS World Congress*, pages 123–134, 2005.

[47] Farzana Rahman, Mehedi Hassan, Alona Martin Kryshchenko, Inna Dubchak, Nikolai Nickolai Alexandrov, and Tatiana Tatarinova. benchNGS : An approach to benchmark short reads alignment tools. *bioRxiv*, page 018234, 2015.

[48] César A. F. De Rose Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov and Rajkumar Buyya. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software - Practice and Experience*, 39(7):701–736, 2009.

[49] Sena Seneviratne and David Levy. Enhanced host load prediction by division of user load signal for grid computing. *Submitted to the Journal of Cluster Computing*, 2005.

[50] Sena Seneviratne and David C. Levy. Task profiling model for load profile prediction. *Future Generation Computer Systems*, 27(3):245–255, 2011.

[51] Sena Seneviratne, David C Levy, and Rajkumar Buyya. A Taxonomy of Performance Prediction Systems in the Parallel and Distributed Computing Grids. *CoRR*, abs/1307.2, 2013.

[52] J Shanthini and K R Shankarkumar. Anatomy Study of Execution Time Predictions in Heterogeneous Systems. *International Journal of Computer Applications*, 45(7):39–43, 2012.

[53] Baiyi Song, Carsten Ernemann, and Ramin Yahyapour. Parallel Computer Workload Modeling with Markov Chains. *Job Scheduling Strategies for Parallel Processing*, 3277/2005: 9–13, 2005.

[54] Tam Mayeshiba. The MAterials Simulation Toolkit (MAST) for Atomistic Modeling of Defects and Diffusion. Technical report, 2016.

[55] Guangming Tan, Chunming Zhang, Wen Tang, Peiheng Zhang, and Ninghui Sun. Accelerating Irregular Computation in Massive Short Reads Mapping on FPGA Co-Processor. *IEEE Transactions on Parallel and Distributed Systems*, 27(5):1253–1264, 2016.

[56] Andrew Thrasher, Douglas Thain, Scott Emrich, Zachary Musgrave, and Ann Arbor. Shifting the Bioinformatics Computing Paradigm: A Case Study in Parallelizing Genome Annotation Using MAKER and Work Queue. In *2nd International Conference on Computational Advances in Bio and medical Sciences (ICCABS)*, pages 1–6, 2012.

[57] Haluk Topcuoglu, Salim Hariri, and M Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *Parallel and Distributed Systems, ...*, 13(3): 260–274, 2002.

[58] Mario Valle, Hannes Schabauer, Christoph Pacher, Heinz Stockinger, Alexandros Stamatakis, Marc Robinson-Rechavi, and Nicolas Salamin. Optimization strategies for fast detection of positive selection on phylogenetic trees. *Bioinformatics*, 30(8):1129–1137, 2014.

[59] V. Taylor W. Smith, I. Foster. Predicting Application Run Times Using Historical Information. *Journal of Parallel and Distributed Computing*, 64(9):1007–1016, 1998.

[60] William A Ward, Carrie L Mahood, and John E West. Scheduling jobs on parallel systems using a relaxed backfill strategy. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 88–102. Springer, 2002.

[61] Tandy Warnow. Large-Scale Multiple Sequence Alignment and Phylogeny Estimation. In *Models and algorithms for genome evolution*, pages 85–146, 2013.

[62] Patrick M. Wensing, Günter Niemeyer, and Jean-Jacques E. Slotine. Performance analysis of a parallel, multi-node pipeline for DNA sequencing. pages 1–11, 2017. URL `http://arxiv.org/abs/1711.03896`.

[63] Marek Wieczorek, Andreas Hoheisel, and Radu Prodan. Towards a general model of the multi-criteria workflow scheduling on the grid. *Future Generation Computer Systems*, 25 (3):237–256, 2009.

[64] Rich Wolski. Experiences with predicting resource performance on-line in computational grid settings. *ACM SIGMETRICS Performance Evaluation Review*, 30(4):41, 2003.

[65] Katherine et al Wolstencroft. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic acids research*, 41 (Web Server issue):557–561, 2013.

[66] Lingyun Yang Lingyun Yang, J.M. Schopf, and I. Foster. Conservative Scheduling: Using Predicted Variance to Improve Scheduling Decisions in Dynamic Environments. *ACM/IEEE SC 2003 Conference (SC'03)*, pages 1–16, 2003.

[67] Jia Yu, Rajkumar Buyya, and Kotagiri Ramamohanarao. 7 Workflow Scheduling Algorithms for Grid Computing. *Meta. for Sched. in Distri. Comp. Envi., SCI*, 146:173–214, 2008.